REPORT DOCUMENTATION PAGE

AD-A265 203

3. REPORT TYPE AND DATES COVERED
FINAL/07 AUG 91 TO 06 SEP 92

4. TITLE AND SUBTITLE
FAST ADAPTIVE MANEUVERING EXPERIMENT (FAME) (U)

5. FUNDING NUMBERS
2305/B3
AFOSR-91-0372

6. AUTHOR(S)
Professor Kenneth J. Hintz

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
George Mason University
Electrical/Computer Engineering
Fairfax VA 22030

8. PERFORMING ORGANIZATION REPORT NUMBER
AFOSR-TR- 93 0269

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
110 DUNCAN AVE, SUTE B115
BOLLING AFB DC 20332-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFOSR-91-0372

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

12b. DISTRIBUTION CODE
UL

13. ABSTRACT *(Maximum 200 words)*

The Fast Adaptive Maneuvering Experiment (FAME) is designed to provide neural network (NN) researchers with a physical, non-linear system of modest dimensionality with coupled dynamics. The system to be controlled is a commerically available model electric helicopter (Whisper) which is secured to a comercially-available stand (Flitemaster, Jr.) which has been modified to limit its range of motion and make it suitable for laboratory operation. The stand has been instrumented with potentiometers to measure all 6 degrees-of-freedom (6-DOF). In order to make the interface to the system as simple as possible a Motorola MC68HC11 microcontroller unit (MCU) has been employed to implement the RS-232 communications protocol, convert the voltages on the potentiometers into angles (8-bit quantization), ,erform the coordinate conversions to a Cartesian space, reply to requests from the NN controller for helicopter position, and translate commands from the NN controller into appropriate servo commands.

14. SUBJECT TERMS

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR(SAME AS REPORT)

Final

Report on the
Fast Adaptive Maneuvering Experiment (FAME)

Sponsored by the

Air Force Office of Scientific Research (AFOSR)

Contract # AFOSR-91-0372

Sponsor: Captain Steve Suddarth, Math and Computer Science, (202)767-5028

March 29, 1992

Kenneth J. Hintz, Ph.D.

Department of Electrical and Computer Engineering

and Center of Excellence in C^3I

George Mason University

Fairfax, VA 22030

(703)993-1592

Report on the
Fast Adaptive Maneuvering Experiment (FAME)

Sponsored by the
Air Force Office of Scientific Research (AFOSR)
Contract # AFOSR-91-0372

Sponsor: Captain Steve Suddarth, Math and Computer Science, (202)767-5028

March 29, 1992

Kenneth J. Hintz, Ph.D.
Department of Electrical and Computer Engineering
and Center of Excellence in C^3I
George Mason University
Fairfax, VA 22030
(703)993-1592

Table of Contents

List of Tables

List of Figures

1. Introduction

The Fast Adaptive Maneuvering Experiment (FAME) is designed to provide neural network (NN) researchers with a physical, non-linear system of modest dimensionality with coupled dynamics. The system to be controlled is a commercially available model electric helicopter (Whisper) which is secured to a commercially-available stand (Flitemaster Jr) which has been modified to limit its range of motion and make it suitable for laboratory operation. The stand has been instrumented with potentiometers to measure all 6 degrees-of-freedom (6-DOF). In order to make the interface to the system as simple as possible, a Motorola MC68HC11 microcontroller unit (MCU) has been employed to implement the RS-232 communications protocol, convert the voltages on the potentiometers into angles (8-bit quantization), perform the coordinate conversions to a Cartesian space, reply to requests from the NN controller for helicopter position, and translate commands from the NN controller into appropriate servo commands.

The source code for all of the software which has been developed for the MCU and the PC communications is provided with the stand so that local modifications can be made as needed. It is in a highly modularized form which allows easy modification once the underlying principle of the software design is understood. A modified commercial MCU board was used (Motorola M68HC11EVB) in order to minimize expenses and provide a limited development/software modification capability to individual researchers. The monitor program on the board has been left intact and can be accessed through the second serial port on the board using Kermit or other terminal emulation software. Actual software development was done using a PC-based C cross-compiler and cross-assembler which supports ANSI C.

The software on the MCU is resident in two forms. The Motorola Buffalo monitor is in its original form on ROM. The board has been modified to incorporate a 32-k byte SRAM with self-contained battery backup (nominal 10 year life). This memory was chosen to enable ease of local software modification as well as easy incorporation of updated software in the field. For all practical purposes, the lower half of memory is populated with non-volatile SRAM in which the program is stored. Conflicts with internal RAM and memory mapped registers are not a problem since the MC68HC11 accesses internal memory when it is available. With this approach, field changes can be made by

modifying the stored program directly through the Buffalo monitor rather than having to erase and reprogram erasable-programmable read only memories (EPROMS).

A calibration program is included as part of the software so that precise mechanical alignment of the stand is not required. If for some reason the stand has been disassembled and the potentiometers moved, this program can be used to realign them in software. The user is prompted to put the movable parts of the stand in certain positions. The MCU then reads the potentiometers, calculates the new offsets and scale factors and stores them in on-chip nonvolatile SRAM. There is no need to recalibrate unless the stand is disassembled or the potentiometers moved from their calibrated positions.

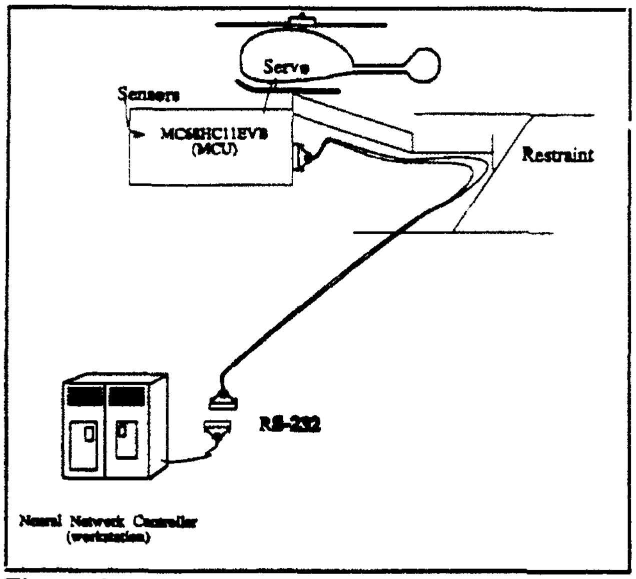

Figure 1 Major system components.

Consideration was given to a number of *safeguards which could be incorporated in the* MCU software, but which have not been implemented. It was felt that any "Deadman's Switch" included in the software would be too intrusive on the desires of NN researchers and limiting on the flexibility of their software design. The hazards of operation of this system should not be taken lightly and, at the very least, appropriate eyeware should be worn when operating the system.

2. Major Components

Three major subsystems comprise FAME: the stand, the helicopter, and the MCU interface. Each of these components has been modified to integrate them into a single system which can be controlled via requests and commands passed to it through an RS-232 interface. The following details the modifications which have been made to each as well as information which may be useful to those who are unfamiliar with model helicopter operations.

2.1. Stand

The stand is a modified commercially available Flitemaster Junior. This stand is usually used by the novice helicopter flyer to learn to hover and maneuver electric or small gas helicopters without the danger of crashing. Although the range of motion is limited, the stand can assist in the development of a pilot's mental framework and the eye-hand coordination necessary for control of this complex system.

The Flitemaster was chosen primarily because it had 6-DOF incorporated in the basic design. Although there are other stands available, this is the only one which has full 6-DOF. As purchased, the stand needs to be modified to make it suitable for NN control of the helicopter. Modifications to the stand are as follows:

1. All axes have been fitted with 5k Ohm linear potentiometers which are supplied with 5 Volts. The potentiometers' outputs are converted by the MCU's A/D converters for angular measurements.
2. A wheel has been added to support the lowest arm to relieve stress on the lowest joint (the center of the H-frame) and improve the accuracy of position measurements.
3. The nylon block support system at the helicopter end of the stand has been replaced with double roller-bearings in order to reduce friction and provide structural support and rigidity to the replacement shaft which supports the helicopter itself.
4. The final nylon-steel ball/socket helicopter support was replaced with an inverted, metal joystick. The potentiometers on this joystick provide roll and pitch angle measurements. A difficulty associated with this support arrangement which has not been resolved is that the center of rotation is below the center of mass of the helicopter and therefore during takeoffs the helicopter platform must be supported to keep it level and stop rotation.
5. Brackets were added to hold the potentiometers which measure joint angles. In some cases, the brackets are also used to limit the range of motion of joints (primarily yaw and the lowest joint at the center of the H). These range limiting mechanisms must not be tampered with or the potentiometers (which do not have a full 360 degrees of motion) will be damaged. Shaft encoders could have been used to allow full range of motion, but this would have been at a significant expense which the budget would not

allow. If potentiometers are removed/replaced/adjusted, care must be taken to insure that they are secured in such a position that the full range of motion of the joint will not damage the potentiometer.

6. The spring support at the middle joint has been modified by changing springs and adding turnbuckles. Modifying the geometry of the springing is ineffective in compensating for the additional weight of the helicopter and MCU. Some compensation is made with the removal of the battery from the helicopter, but stronger springs are needed to mass balance the system.

7. Wiring has been added to provide electrical power to the MCU, power to the electric helicopter motor, and carry the joint potentiometer signals to the MCU's A/D for conversion.

The stand should be secured to the floor and an area cleared to insure unimpeded flight of the helicopter. If a permanent mounting cannot be made, two or more 25 lb bags of lead shot have been used to keep the H-frame immobile while the helicopter is flows.

2.2 Kalt Whisper Helicopter

The helicopter is manufactured by Kalt and is commercially available as the Whisper. The helicopter and spare parts are available at local hobby shops. Because the Whisper utilizes an electric motor as power for the main rotor it must carry a nickel-cadmium (NICAD) battery on board for power. In order to carry this load, the frame is very light and therefore not very strong. It will not take much abuse. In particular, the ball links which actuate the controls are particularly easy to break and the use of ball-link pliers for disassembly is strongly encouraged.

The Whisper kits were purchased in almost-ready-to-fly (ARF) form. Experience has shown that the modest difference in price between the kit and the ARF kit is well worth the savings in construction time. The ARF kits have been completed and the helicopter adjusted and aligned. The helicopter construction manuals are delivered with FAME, but alignment is a tedious process and should not be necessary except in the event of mishap. In particular, do not remove the tape from the main rotor blade(s). This tape is used to balance the rotors as well as a sighting device when the blades are

aligned for proper tracking (both blades following the same path). Perfect tracking is difficult to obtain, and the systems are adjusted as close as possible when delivered.

There are two switches on the plate which supports the helicopter. The on/off switch supplies power to the electric speed controller. This switch is used in normal helicopter installations with radios to provide a regulated voltage to the radio receiver. There is a second switch, a push-button, which enables the speed controller to supply electricity to the motor. There are two switches in the normal hobby installation so that the radio receiver can be enabled and a check made to insure that the throttle is at such a setting that the motor will not turn when the motor is enabled. If power is not supplied to the MCU when the push button is enabled, unpredictable results will occur (the motor could start turning and hit the person actuating the button). Insure that the MCU is operating and that the green light is lit on the motor controller before pushing the button. The initial setting of the throttle by the MCU is at its minimum value.

2.2.1. SERVOS

Ball bearing servos are used throughout to insure long life and reliability. The particular servos used are Futaba FPS-133. While the operation of the servos is transparent to the users, a brief explanation of the electrical mode of actuation follows. Each servo is controlled by a 5 Volt pulse of variable width. A pulse width of 1.0 ms positions the servo at an extreme end of its rotation. A pulse width of 2.0 ms positions the servo at the other extreme end of its rotation. The nominal zero for the servo occurs at a pulse width of 1.5 ms. The angle between these two extremes is proportional to the pulse width between 1.0 and 2.0 ms. The exact values are not given here since there are servo-servo differences and the linkage is not necessarily linear between the servo and the control actuated (usually rotary to linear motion conversion is involved with a limited range of linear operation). That is, the resultant amount of control action is not necessarily linearly related to the servo command.

The servo **does not respond** to a single pulse width command, but the command must be repeated at regular **intervals to insure** complete motion to the commanded angle. In order to relieve the NN controller of **this burden**, the MCU interface continuously sends the commanded pulse width to each servo. The software is designed such that a two bytes control word covers the complete range of motion of each servo. That is, the pulse width in units of 500 ns/unit are transmitted as unsigned

integers (e.g., 2000 decimal = 1.0 ms). The actual pulse width is generated using the output compares of the MC68HC11 and is a background process under the control of interrupts.

One servo is not mechanical and that is the speed controller. The format of the control signals for it is the same as the other servos, but it is solid-state and has no moving parts. This improves reliability as well as accuracy of control over other methods which have a mechanical servo controlling the position of a high current potentiometer. With this method, there is little wasted energy since the SS speed controller is operated in a switching mode. Normal model helicopter have a mixing function incorporated either on the helicopter or in the transmitter. This mixing function couples the throttle (SS speed controller) to the collective pitch. These are not coupled in the MCU implementation and are completely separate channels available for individual control.

2.2.2. GYRO

There is one single-rate Futaba gyro mounted on the helicopter for yaw stabilization. The normal configuration for a yaw rate gyro is to have the tail rotor signal from the receiver (the 1.0 to 2.0 ms pulse) connected to the gyro. The gyro then modifies this pulse width according to yaw-dot to generate a yaw disturbance negative feedback signal. The Futaba model number of this special gyro for electric helicopters is Futaba G-155.

In the FAME configuration, the yaw rate gyro is driven by a 1.5 ms pulse from the MCU in response to a position measurement command from the workstation. The pulse-width gyro output is then read by the MCU and converted to an eight-bit signed value. This value can then be returned to the workstation along with the 6-DOF coordinates. Although the electrical connections are made on FAME, the software does not yet implement this function.

2.3. M68HC11 EVB

Motorola sells an evaluation board based on its MC68HC11 microcontroller (MCU). This board was selected because of its low cost and inclusion of the port replacement unit (PRU). The PRU allows the MCU to be operated in the expanded mode (one of its four modes of operation) while retaining all of the original ports as if it were operating in the single-chip mode. Operation in the expanded mode

allows for the virtual memory (64 kbytes) to be fully populated with either RAM or ROM. The additional ROM is required because of an early decision to write as much code as possible in C in order to make it easier for the user to modify it to suit his particular needs. The printing and scanning functions in particular require a lot of space. These printing functions are primarily used when the optional monitor is attached and operating in a terminal mode.

An additional reason for using the evaluation board is that it comes with complete circuit diagrams as well as a complete hardware reference manual. Also bundled with it is a bare-bones cross-assembler (non-macro) which could be used in a pinch to make necessary modifications to the MCU code.

The board as it comes from the factory is stuffed with 8 kbytes of RAM at address $C000 and this is retained. A SRAM of 32 kbytes is added to insure sufficient space for a high-level language program. The SRAM chosen has a built-in battery backup so that it retains its memory even when power is removed. The SRAM acts as if it were an EEPROM except that it is much easier to modify in that it only needs to be written to just like any other RAM. While EEPROM has a sufficiently short access time during operation, it has an unacceptably long programming time. Minor modifications are made to the board and the socket for an optional 8 kbyte RAM ($6000) is used to hold the 32 kbyte SRAM. The following modifications are required to utilize the Dallas DS1230Y-150 32kx8 nonvolatile SRAM on the M68HC11EVB:

Table I Jumper settings for M68HC11EVB.

Jumper Number	Setting
1	Open
2	2-3
3	Open
4	1-2 (Buffalo) 2-3 (FAMEmain)
5	9600
6	Closed
7	Open

1. Replace R2 (10 kOhm) with approximately 3.3k Ohm. Smaller values will cause the M68HC11 to sink too much current. Larger values prevent the passive pullup from charging the input to the chip sufficiently fast. Occasional errors occur in memory if this is not done.

2. Disconnect Jumper J3 to disconnect chip select from chip select decoder. Address bit 15 is used as CS~ to select the ·er half of memory.
3. Cut the trace to pin 11/U12E ef...tively removing R/W* from pin 11.
4. Connect pin 11/U12E to pin 1/U12A effectively supplying the E-clock to pin 11
5. Connect pin 20/U4 to pin35 of U10. This connects A15 to CS~ of the SRAM
6. Cut the trace between pin 26/U4 and pin 28/U4.
7. Connect pin 26/U4 to pin 37/U10. This connects A13 to the SRAM.
8. Connect pin 1/U4 to pin 36/U10. This connects A14 to the SRAM.

The normal 8kx8 SRAM at $C000 (U5) can be retained and is needed to run the calibrate program The addition of the 32kx8 effectively fully populates the lower half of M68HC11 memory. Those memory addresses which are internal to the M68HC11 remain so and there is no conflict between the external SRAM a. internal locations. The memory between $4000 and $5FFF may be unusable depending on whether you program modifies the SCI control flip-flop, U11B.

As delivered, the jumpers on the MC68HC11EVB board should be as listed in Table I.

All signal interconnections are made to the board through the 60 pin connector. Power (0 V, +5 V) is supplied to the 6-DOF potentiometers from the MCU board through an eight position Dean's connector female block. Ground and signal lines are supplied to the helicopter servos through another block. Plus 5 Volts is not supplied from the MCU board since excessive current drain during servo actuation causes the board to malfunction. Power is supplied to the servos from the speed controller servo (SCS) which has an internal regulator. The SCS regulates its +12 Volt supply voltage down to +5 volts and normally powers the receiver and servos. In FAME, this regulated +5 Volts is distributed to the servos through the servo connector block. A common ground is provided between the MCU and the SCS.

If it is desired to have an external restart button, pin 17 of the 60-pin connector can be extended off the stand along with a ground wire. Grounding pin 17 of the 60-pin connector will restart the board as long as jumper J1 is in place.

2.3.1. Buffalo Monitor

As delivered from Motorola, the evaluation board has a resident monitor called Buffalo. This monitor has a limited repertoire and is designed for hands on experimentation at a simple level. Since the monitor does allow some direct access to the MCU as well as the ability to download Motorola S-record format files, it has been maintained. Whether to restart in Buffalo or in the normal FAME operating program is determined by jumper J4. If connected between pins 1 and 2 restart causes Buffalo to come up. If J4 is connected between pins 2 and 3, then the FAMEMAIN program begins to execute after pushing the restart button.

Documentation for Buffalo is found in the evaluation board manual. When uploading S-record format files using Kermit, be sure to include the \0 on the transmit command so handshaking protocol is eliminated, e.g., "Transmit FAMEmain.0 \0". This does not appear to be a problem using PROCOMM in the ASCII file upload mode. Note that one uploads a file rather than downloads it to the MCU. Due to differences in the way operating system BIOSs handle CR/LF, FAME software was compiled with only \r rather than \n\r. If lines overwrite themselves, enable a switch on your terminal program which converts CRs to CR/LF.

2.3.2. Power Supply

Two power supplies are required for operation of FAME. One is supplied, the other must be locally obtained. The one supplied is a used IBM PC, 63 Watt power supply which is sufficient to power the MCU board and has the requisite +12, -12, and +5 Volt taps. The second power supply which is required is 9.6 to 12 Volts for powering the helicopter motor. A circuit diagram of one used at Drexel University is included in the appendix. A 9.6 Volt, 1000 mAh NICAD battery is provided to get you started, but will only supply enough current for 10-15 minutes of operation. These are the normal flight batteries. This battery should be recharged at the normal c/10 rate of 100 mA. Quick charging is possible if care is taken to not overcharge the batteries. Constant voltage charge is NOT an acceptable method for recharging NICAD batteries. Twelve Volt Gel-Cell batteries are recommended if a sufficiently robust power supply cannot be obtained. Conventional wet-cell lead acid batteries can be used if proper ventilation is maintained during recharging to prevent hydrogen buildup. Normal current draw for the motor is in excess of 10 Amperes.

The power for the servos is supplied through the electronic speed controller, so, in order for the servos to function under 68HC11 control, power must be supplied to the speed controller and it must be switched on. As long as the push-button is not pushed, power will not be supplied to the motor, although the green and red lights on the speed controller will change as it is still receiving commands. The +5 from the speed controller and the 68HC11EVB are not tied together. There is a common ground reference between the two so that the signals from the EVB can control the servos.

3. Operational Interface

The basic communication principle employed between the users workstation and the MCU is that the MCU responds to commands passed to it from the workstation over an RS-232 bidirectional connection operating at 9600 baud. Commands include servo control data and position requests among others. These commands/requests are in the form of a serial byte string comprising a start character, a control character, one or more data bytes, a checksum, and a stop character. The message format must be exact or it is rejected. The user's workstation software should check for a response from the MCU within a short time to insure that has received the message and is working properly. The accompanying PC based software incorporates such checking.

3.1. MC68HC11 Communications

There are two serial communication ports on the M68HC11EVB. The one which is internal to the MCU is used for communications with the workstation and is under the direct control of the FAMEMAIN.C program. The other serial port is handled by an off-chip 8250 UART and is used to communicate to a terminal (usually a PC running a terminal emulation program such as Kermit or Procomm). In normal operation, the terminal does not need to be connected and is only necessary for using the BUFFALO monitor, downloading modified programs, or using special testing software.

Serial communications to/from the workstation are handled in the MCU through an interrupt driven Serial Communications Interface (SCI) routine. The complete message is stored character by character as it is received and the checksum and format verified before a semaphore is set indicating the reception of a complete message. When the foreground detects this message received semaphore, it enables other foreground and background functions to take place by setting or clearing other

semaphores. The dichotomy between foreground/background emphasizes the architectural uniqueness of the MCU in that many functions can be performed in hardware with interrupts indicating their completion. Extensive use is made of this feature so that TX/RX communications can be maintained in the background while interruptable foreground processes continue.

Table II Signal description for 6-pin Molex connector supplying power to stand.

Pin Number	Description	Color Code
1	Motor Gnd	Thick Black (#12)
2	Motor +12V	Thick Red (#12)
3	6811 Gnd	Black
4	6811 +5V	Brown
5	6811 -12V	Red
6	6811 +12V	Orange

3.2. Workstation/MC68HC11 Message Formats

There are several messages which can be sent between the workstation and the MCU. The start character is ASCII "S" and the stop character is ASCII "s". The checksum is the modulo 256 sum of all data except the start character, the checksum byte and the stop character. The position variables are all 16-bit integers and the most significant byte (MSB) must be put in PostTXbuff[i] and the least significant byte in PosTXbuff[i+1]. Data are not encoded as characters, but are the actual binary values which must be concatenated. The number in parenthesis is the position of the character or byte in the message string. The available messages are listed in Table III.

3.3. Workstation (PC) Communications

In order to verify proper software operation, a communications protocol interface is supplied which is PC based If a PC is not available, then the C source can be recompiled for alternative machines. The program is not a control program, but rather the skeleton of a communications routine which would be included as part of a neural network control implementation. Of particular interest to PC users is the interrupt service routine for serial communications. MS-DOS does not use an ISR for serial communications and is therefore not suitable for MCU communications. This difficulty is circumvented by installing an ISR to handle serial communications. There appears to be incompatibilities among PC hardware and the software is not guaranteed to work on all PCs. It was

Table III Workstation/MCU Message formats.

ORIGINATOR	MESSAGE	MESSAGE STRING
Workstation	Request Position Values	Start Char(0), R(1), Stop Char(2)
Workstation	Command Servo Control	Start Char(0), C(1), Servo Control Values(2-11), Checksum(12), Stop Char(13)
Workstation	Request Potentiometer Values (not yet implemented)	Start Char(0), ___(1), Stop Char(2)
MCU	Send Position Values	Start Char(0), P(1), X(2-3), Y(4-5), Z(6-7), Roll(8-9), Pitch(10-11), Yaw(12-13), Checksum(14), Stop Char(15)
MCU	Send Acknowledgement of receipt of control values	Start Ch.), A(1), Stop Char(2)
MCU	Send Potentiometer Values (not yet implemented)	Start Char(0), ____(1), H-Pot(2-3), Az-Pot(4-5), El-Pot(6-7), Roll Pot(8-9), Pitch Pot(10-11), Yaw Pot(12-13), Checksum(14), Stop Char(15)

developed using Borland C++ and ran successfully on an ATT-6300 (286) and an INTRA LT-386sx laptop (it did not run on a Zenith 386/16).

3.4. Coordinate Conversion

The measurements of the 6-DOF are not made in Cartesian space but rather in joint space. These angular measurements are converted by the MCU into an X, Y, Z, roll, pitch, and yaw format. This is done by applying the calibration values which are stored in nonvolatile RAM to the measurements and then computing the Cartesian coordinates through trigonometric conversions. The roll, pitch, and yaw do not require coordinate conversions since they are measured directly. A scale factor stored in nonvolatile RAM is still applied to convert the measured voltages into angles. Units for these measurements are centimeters for X, Y, and Z and degrees for Roll, Pitch, and Yaw. Scaled values are transmitted to maintain resolution. The current version of the software (1.0) scales all values by a factor of 100 before transmitting them back to the PC.

3.5. Servo Outputs

In normal radio controlled operation, servos expect to receive their 1.0 to 2.0 ms signal pulse once per 1600 sec. Although the interval does not have to be this long, they will malfunction with too short an interval. As there was not enough output compare to control all the servos, each output compare controls 1 pin PA and 1 pin PB.

Table IV Servo block connections.

OUTPUT COMPARE NUMBER	SERVO BLOCK CONNECTOR NUMBER	PORT NUMBER	MCU PIN NUMBER	SERVOS CONTROLLED
OC3	6	PA5	29	Collective
OC3	5	PB2	40	To Gyro
OC4	4	PA4	30	Rudder
OC4	3	PB1	41	Elevator
OC5	2	PA3	31	Aileron
OC5	1	PB0	42	Throttle
IC1	0	PA2	32	From Gyro

The servo connections are organized as shown in Table IV:

3.6 A/D Input Block

The six potentiometers which measure the angles of the six joints are connected to the MCU through a multi-connector block immediately behind the M68HC11EVB. As with the servo connections, the wires are numbered with the corresponding number on the block. Power is supplied to the A/D block from the V_{refH} and V_{refL} of the M68HC11EVB. The A/D channels and connector block pin assignments are as in the following table.

3.7. System Verification Software

A PC based program, FAMEPC.C, is provided which demonstrates the control capability available

Table V A/D converter signal/pin assignments.

A/D CHANNEL	A/D REGISTER	PORT NUMBER	M68HC11EVB PIN NUMBER	A/D BLOCK PIN NUMBER	SIGNAL
1	ADR1	PE0	43	-	Buffalo
2	ADR2	PE1	45	1	Roll (3Y)
3	ADR3	PE2	47	3	Yaw (3Z)
4	ADR4	PE3	49	5	Pitch (3X)
5	ADR1	PE4	44	0	Az(2Z)
6	ADR2	PE5	46	2	El(2X)
7	ADR3	PE6	48	4	H (1Z)
8	ADR4	PE7	50	6	-

through the RS-232 interface to the MCU. Servo position values can be typed in directly in milliseconds of control value (from 1.0 to 2.0 ms) or the individual servos increased or decreased by their smallest value through pushing the upper case value of the controlling letter or the lower case letter respectively. The controlling letters are shown in Table VI.

Table VI PC keyboard controls.

SERVO	CONTROLLING (UC INCREASES, LC D
Throttle	T o
Tail Rotor	R o
Collective	C o
Elevator	E o
Aileron	A o

3.8. Calibration

In order to eliminate tedious alignment of potentiometers at each joint, it was decided to align the joints electrically rather than mechanically. That is, the exact position of the potentiometer is not important as long as it can be corrected prior to its use in the conversion Cartesian coordinate conversion. This is done by measuring the voltages produced at specified angles during a calibration program and computing a zero and scale factor. These factors are stored in nonvolatile RAM. As

long as the stand is not disassembled, there is no need to recalibrate since the correction values are stored in nonvolatile storage which is retained even if power is removed.

The calibration program, FAMECAL.C, must be downloaded into the M68HC11EVB and started using the Buffalo monitor. A terminal emulator must be connected through the port other than the one which is connected to the workstation. Communications are at 9600 baud, no parity, one stop bit. If power is removed from the evaluation board, the program will be lost since part of it is loaded into volatile RAM. This is not the case for FAMEMAIN since it all resides in non-volatile RAM.

3.9 DOF Range Limitation

There is a red colored round collar immediately below the platform which holds the helicopter. That ring is fixed in position with screws. The collar is there to limit the range of motion in the roll, pitch, and yaw axes during initial NN control program development. The total range of motion about these axes is great enough with the red collar removed that the tail rotor and boom can strike the stand and cause extensive damage to the helicopter. After some confidence is gained in the control algorithms and its implementation, the red collar can be lowered by removing the two screws and drilling and tapping new holes at a lower position. The tap size is 6-32.

An easier way to lower the collar is to compress the springs by tightening the bolts which hold the two red rings together. The best alternative which will allow complete range of motion and yet not require the removal of the platform/helicopter as detailed below is to remove the four bolts/springs which couple the two and then use shorter bolts/nuts to hold the two red rings together with no distance between them.

Complete removal of the collar requires that the helicopter and platform be removed from the supporting shaft. This could be done either of two ways. The recommended removal technique is to disconnect all of the wires from the helicopter/platform to the servo and A/D blocks as well as the motor power supply. Slowly rotate the helicopter/platform counter clockwise (CCW) when viewed from above until the allen head bolt which limits the range of motion of the yaw axis makes contact with the bracket at the bottom of the platform supporting shaft. Continue turning the platform with steady CCW pressure until the threaded portion of the shaft disengages from the inverted joystick on

the platform (the joystick measures roll and pitch angles). It will take several turns to remove the platform from the supporting shaft. The washers are required between the shaft and the joystick to compensate for the fact that the threads could not be machined completely to the flat face of the shaft and it is the flat face of the shaft which must mate with the flat surface of the joystick to produce true rotational motion. The number of washers determines the zero yaw position of the helicopter. Some washers must be present when reassembling the platform/shaft. Be sure to use Locktite on the threads when reassembling and do not overtighten as there is a chance of stripping the threads in the aluminum joystick (they are quite expensive) or shearing the threaded portion of the supporting shaft (which is custom made).

A less desirable alternative is to disconnect all wires as above and then unscrew the allen head cap screw which connects the supporting shaft to the yaw axis potentiometer at the bottom of the shaft. If the spring clip immediately above the allen head capscrew is then removed, the shaft should be able to be pulled upwards through the two roller bearings, effectively removing the shaft, platform, and the helicopter simultaneously. This method is **not** recommended as the alignment of the bearings is critical for smooth operation about the yaw axis and, more importantly, the yaw axis potentiometer must the be recalibrated both electrically and mechanically. Electrical alignment is required so that the proper angle is measured by the A/D converter. Mechanical alignment is required to insure that the allen head capscrew limits the range of motion about the yaw axis so that the potentiometer is not destroyed. The potentiometer is not strong enough in rotation to stop the helicopter and a mechanical stop must be used.

There is a second purpose for the red collar. A close inspection will reveal that one of the screw/nut pairs through the springs is reversed which causes the nut to protrude slightly above the red ring. This is done so that the helicopter platform has something to push against while the rotor speeds up and before the tail rotor has reached an effective speed for controlling yaw. Once the helicopter is hovering, the platform does not make contact with the nut. The commercial speed controller which is supplied does not have a slow ramp up and tends to jerk the main rotor even with minimal control action. This too can cause the helicopter to yaw violently leading to the possibility of damage.

3.10 Connector Wiring

The standard connector for wir.ng signals is the "Dean's" connector. This is a three pin connector which we have used in the following standard. One end of the connector is distinguished by a groove between two pints. Starting at this end, the signals on the connector are: Signal, +5 Volts, ground. This is true for the potentiometers used to measure the angles as well as the servos. It is possible to connect these connectors backwards in spite of the fact that there is a different spacing between the signal and +5V pin. The notches must line up. On connecting the Dean's connectors to the servo and A/D block, note that the block has a white line which indicates the position of the groove in the connector.

Table VII Signal description for 4-pin Molex connector supplying power to MC68HC11EVB

Pin	Signal	Color
1	Ground	Black
2	+5 Volts	Brown
3	-12 Volts	Red
4	+12 Volts	Orange

Exercise caution when wiring around the servo and A/D blocks which are epoxied to the stand behind the MC68HC11EVB as the pins which protrude from the blocks (next to the connectors which are plugged in) have +5 Volts and Ground on the lower two pins .

The system has been supplied with an 6-pin Molex connector through which power connections are made to the system from the external motor power supply and the PC power supply. The voltages on these pins are as listed in Table VII. Female pins are used on the power supply side and male pins used on the stand side.

A four pin Molex connector is used to connect power to the MC68HC11EVB. The voltages on these pins are as listed in Table VII.

4. Software Development

All of the MCU software was developed using a C cross-compiler/cross-assembler from INTROL

corporation. If further development by the user is desired this, or a similar cross-compiler/cross-assembler, needs to be purchased. All of the source code for both the PC and the MCU is available on disk as well as listed in the appendices The emphasis was on C with only those portions of code requiring the fastest processing or unusual hardware control written in assembly language

PC based software for the calibration program and sending/receiving of commands, requests, and messages was developed using Borland Turbo C++, although no ++ enhancements were used and the program was compiled with Turbo keywords on so that PC specific I/O routines could be used.

4.1. Foreground/Background Software Design

The implementation of the code is best understood with reference to a foreground/background approach. Interruptable processes are run in the foreground with their execution and behavior controlled by tokens which are either set in other processes or in the interrupt service routines. Some tokens are hardware flags which are read to determine their TRUE/FALSE value. For example, the serial communications interface (SCI) in the MCU is interrupt driven. Upon receipt of a character, the interrupt service routine (ISR) first checks to see if there are any errors (overrun or framing), then checks for start character and/or command character, stores data characters and then compares the locally generated checksum with the one transmitted. Only when all of these checks are complete in the background ISR is a semaphore set which tells the foreground process that a complete message has been received.

Hardware background processes are used extensively because they are readily available on the 6811 MCU. For example, the pulses sent to the servos are controlled through the use of the output compare (OC) registers in the MCU. These registers are loaded with a value. When the value in the OC register matches that of the internal free-running counter (TCNT), an interrupt is generated. Since there are only 5 OCs, three of them are used to drive two servos. The ISR for an OC determines which servos pulse width has passed and clears it to zero before reloading the OC register with a value that will tell it when to turn off the other servo which it controls. It must be emphasized that these OCs are hardware compares which operate in the background and generate an interrupt when their value matches that of the free-running counter. After both pulses are cleared to a low value, the array of pulse widths is read to determine the duration of the low value and this is timed. After the low

value interrupt occurs, a new round of high pulses is programmed. The OCs are loaded from the arrays which store the values. It is the foreground process which, after detecting a message received semaphore, reads the command message and stores the data in the array of pulse width duration values. Since the actual PWs are generated in the background continually, the new pulse widths are output as soon as they arrive. Output pulses are generated sequentially, the automatic bit set feature of the OC was not used.

4.2. M68HC11EVB Buffalo Monitor

The Motorola supplied Buffalo is maintained intact in the evaluation board. Its limited repertoire is still usable for unassembling or modifying the executable code. On restart, the MCU fetches the address of the startup routine from locations $FFFE/FFFF. As delivered in the 68HC11EVBU, this points to the Buffalo monitor. In order to start in either Buffalo or FAME, the user must set jumper J-4 to connect pins 2-3 rather than pins 1-2 for Buffalo. The Buffalo monitor reads this jumper to determine which mode of operation is desired. For users without a PC, this allows using FAME without always having to start up in the Buffalo monitor and then switch to FAME through the GO command.

4.3. INTROL C Cross-Compiler/Cross-Assembler

To one familiar with Borland C++, the INTROL cross-compiler is a bit difficult to use. The error messages are sometimes cryptic, and the linker command file (which controls the object module linking process) is difficult to set up. To minimize confusion about the method of locating the ISRs and their pointers during the linking process, the linker command file is listed in section 6.3, Appendix III.

5. System Support

Questions concerning FAME can be directed by email to khintz@fame gmu.edu. Very limited repair parts are available from GMU and the user is expected to become self-sufficient once delivery of FAME is made. To assist in that end, the next section lists suppliers of the various components. Dimensioned drawings of custom mechanical parts are available and included in the appendix.

Also available for 1 week loan are two VHS video tapes which are a basic introduction to model helicopter construction and flying. The two titles which we have from the 'Milt Video Library' are "Model Helicopter Building and Flying Techniques", and "Advanced Helo Flight Course, Intermediate and Advanced Techniques." Don't rush out and buy them, but they are worth borrowing from us.

5.1. Spare Parts (suppliers)

Ace R/C Inc.
116 W. 19th Street
P. O. Box 511
Higginsville, MO 64037
(816)584-7121
FAX (816)584-7766

Radio control electronic parts, Deans Connectors

Helicopter World, Inc.
521 Sinclair Frontage Road
Milpitas, CA 92035
(408)942-9521
FAX (408)942-9524

Kalt Whisper Helicopter
Flitemaster Junior Stand
Futaba G155 Gyros
Futaba S133 Servos

Miniature Aircraft USA
2324 N. Orange Blossom Trail
Orlando, FL 32804-4896
(407)422-1531

X-Cell Gas Helicopters (not used in electric FAME)

Motorola

Microprocessor Products Group

6501 William Cannon Drive West

Austin, Texas 78710

Design Kit, 68HC11EVB

Sterling Electronics

6304 Woodside Ct.

Columbia, MD

1-800-767-7176

Dallas Semiconductor 256 kbyte nonvolatile SRAM #DS1230Y-150ns

Digikey Corp

701 Brooks Avenue South

P. O. Box 677

Thief River Falls, MN 56701-0677

1-800-344-4539

Molex Connectors #03-09-204 (4 pin plug), #03-09-1041 (4 pin receptacle)

ASC 60G-ND Socket Connector, 60 pin, gold

ASSR60-ND 60 pin strain reliefs

Introl Corporation

647 W. Virginia Street

Milwaukee, WI53204

(414)276-2937

fax (414)276-7026

M68HC11 C cross-compiler and cross assembler

TTI Inc.

6420-B Dobbin Dr.

Columbia, MD 21045

(301)995-1331

Potentiometers, RV6NAYSD502A, Type W, 5kOhms, Linear Taper, single turn

5.2. Acknowledgements

Instrumental in the development of FAME have been Bertina Ho-Mock-Qai, a visiting research scholar, Elsa Lam, a senior Electrical Engineering Student, and Darrell Duane, a graduate research assistant in Electrical Engineering.

6. Appendices

6.1. Appendix I: MC68HC11 C Source Code

This code is included for reference only. It is a fully functional, correctly operating program but does not contain the latest updates. Please contact khintz@fame.gmu.edu for the most recent version of the code.

```
/*-----------------------------------------------------------------------*
/*                       George Mason University                          *
/*         Department of Electrical and Computer Engineering              *
/*                                                                         *
/* File name:  FAMEDEF.h                                                   *
/*                                                                         *
/*  Authors: Bertina Ho-Mock-Qai, Darrell Duane,  Ken Hintz                *
/*  Update History:  Version 1.0, February 23, 1992                        *
/*                                                                         */
/*    Header file for FAME operating program in M68HC11                    */
/*                                                                         */
/*-----------------------------------------------------------------------*/

#ifndef FALSE
#define FALSE 0
#define TRUE !FALSE
#endif

/*-----------------------------------------------------------------------*/
/*      masks used for bitwise operations on registers or variables      */
/*-----------------------------------------------------------------------*/

#define MASK0 0xFE          /* 1111 1110 */
#define MASK1 0xFD          /* 1111 1101 */
#define MASK2 0xFB          /* 1111 1011 */
#define MASK3 0xF7          /* 1111 0111 */
#define MASK4 0xEF          /* 1110 1111 */
#define MASK5 0xDF          /* 1101 1111 */
#define MASK6 0xBF          /* 1011 1111 */
#define MASK7 0x7F          /* 0111 1111 */

#define CMASK0 ~MASK0       /* 0000 0001 */
#define CMASK1 ~MASK1       /* 0000 0010 */
#define CMASK2 ~MASK2       /* 0000 0100 */
#define CMASK3 ~MASK3       /* 0000 1000 */
#define CMASK4 ~MASK4       /* 0001 0000 */
#define CMASK5 ~MASK5       /* 0010 0000 */
#define CMASK6 ~MASK6       /* 0100 0000 */
#define CMASK7 ~MASK7       /* 1000 0000 */

#define IC1_NUM    0        /* number of input capture 1 */
#define IC2_NUM    1        /* number of input capture 2 */
#define IC3_NUM    2        /* number of input capture 3 */
#define OC1_NUM    0        /* number of OC capture 1    */
#define OC2_NUM    1        /* number of OC capture 1    */
#define OC3_NUM    2        /* number of OC capture 1    */
#define OC4_NUM    3        /* number of OC capture 1    */
#define OC5_NUM    4        /* number of OC capture 1    */
#define PA3        3        /* pin number on port A      */
```

```
#define PA4        4         /* pin number on port A      */
#define PA5        5         /* pin number on port A      */
#define PA6        6         /* pin number on port A      */
#define PB0        0         /* pin number on port A      */
#define PB1        1         /* pin number on port A      */
#define PB2        2         /* pin number on port A      */

#define A_LOW       0        /* indicates status of servo outputs */
#define A_HIGH      1
#define B_LOW       2
#define B_HIGH      3

#define PI  3.14159265359    /* value for pi */
#define RADIANS_TO_DEGREES ((double)180/(double)PI) /*conversion  */
#define ANGLE_TX_SCALE_FACTOR 100 /* scale for tx angle integers  */
#define POSITION_TX_SCALE_FACTOR 100 /* scale for tx pos integers */

#define ANGLE_1z0 0          /* calibration location 0 for pot 1z */
#define ANGLE_1z1 (PI)
#define ANGLE_2x0 0
#define ANGLE_2x1 (-PiOverFour)
#define ANGLE_2z0 0
#define ANGLE_2z1 (PiOverTwo)
#define ANGLE_3pitch0 0
#define ANGLE_3pitch1 N/A   /* this value prompted for from user */
#define ANGLE_3roll0 0
#define ANGLE_3roll1 N/A   /* this value prompted for from user */
#define ANGLE_3yaw0 0
#define ANGLE_3yaw1 (PiOverTwo)
#define ATOD_ERROR_LIMIT 10  /* sum of max differences in four A/D samples */

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/* defines for initializing RAM ISR jump table                                */
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
#define JUMPEXTENDED 0x7E    /*Assembly language inst. for ISR jump table*/
#define VSCI    0x00C4       /* Serial Communications Interface          */
#define VSPI    0X00C7       /* Serial Peripheral Interface              */
#define VPAIE   0X00CA       /* Pulse Accumulator                        */
#define VPAO    0X00CD       /*                                          */
#define VTOF    0X00D0       /* Timer Overflow                           */
#define VTOC5   0X00D3       /* Output Compare 5                         */
#define VTOC4   0X00D6       /* Output Compare 4                         */
#define VTOC3   0X00D9       /* Output Compare 3                         */
#define VTOC2   0X00DC       /* Output Compare 2                         */
#define VTOC1   0X00DF       /* Output Compare 1                         */
#define VTIC3   0X00E2       /* Input Capture  3                         */
#define VTIC2   0X00E5       /* Input Capture  2                         */
#define VTIC1   0X00E8       /* Input Capture  1                         */
#define VRTI    0X00EB       /* Real Time Interrupt                      */
#define VIRQ    0X00EE       /* Maskable Interrupt Request               */
#define VXIRQ   0X00F1       /* Non-Maskable Interrupt Request           */
#define VSWI    0X00F4       /* Software Interrupt                       */
#define VILLOP  0X00F7       /* Illegal Operation                        */
#define VCOP    0X00FA       /* Computer Operating Properly              */
#define VCLM    0X00FD       /*                                          */
#define VRST    $E000        /* Restart Buffalo Monitor using assembly   */
```

```
/*-----------------------------------------------------------------------*
/*                     Definitions for Handshaking                       *
/*-----------------------------------------------------------------------*

/* Disable TX data buffer empty interrupt                                *
#define DisableTXbuffEmptyInt() ClearBit(SCCR2_Add,7)
/* Enable TX data buffer empty interrupt                                 *
#define EnableTXbuffEmptyInt()  SetBit(SCCR2_Add,7)
/* Disable TX complete interrupt                                         *
#define DisableTXcompleteInt()  ClearBit(SCCR2_Add,6)
/* Enable TX complete interrupt                                          *
#define EnableTXcompleteInt()   SetBit(SCCR2_Add,6)
/* Disable RX start interrupt                                            *
#define DisableRXint()          ClearBit(SCCR2_Add,5)
/* Enable RX start interrupt                                             *
#define EnableRXint()           SetBit(SCCR2_Add,5)
/* Disable idle line interrupt                                           *
#define DisableIdleInt()        ClearBit(SCCR2_Add,4)
/* Enable idle line interrupt                                            *
#define EnableIdleInt()         SetBit(SCCR2_Add,4)

#define START_CHAR (unsigned char) 'S'                /*0x53 upper case S */
#define STOP_CHAR  (unsigned char) 's'                /*0x73 lower case s */

/* Command Char RXed by HC11 that initiates calc. & TX of posit. values  */
#define POS_REQ_COM_CHAR (unsigned char) 'R'
/* Command Char RXed by HC11 that preceeds servo control values          */
#define SER_REQ_COM_CHAR 'C'
/* Command Char TXed by HC11 that preceeds position values               */
#define POS_ACK_COM_CHAR 'P'
/* Command Char TXed by HC11 that ack receipt of servo control values    */
#define SER_ACK_COM_CHAR 'A'
/* Command not yet implemented*/
#define BAD_REQ_COM_CHAR 'x'

#define START_CHAR_INDEX 0
#define COM_CHAR_INDEX 1

/* 12 position bytes = 6 values + 4 handshaking chars                    */
#define POS_ACK_STRING_LENGTH 16
/* 1 start char, 1 command char, & 1 stop char                           */
#define SER_ACK_STRING_LENGTH 3
/* 1 start char, 1 command char, & 1 stop char                           *
#define POS_REQ_STRING_LENGTH 3
/* 10 servo control bytes = 5 values + 4 handshaking chars               */
#define SER_REQ_STRING_LENGTH 14

/* number of points (0 included) of position data                        */
#define POS_ACK_DATA_ONLY 11
/* number of points (0 included) of servo control data                   */
#define SER_REQ_DATA_ONLY 9

#define X_MSB     2  /* index of pos reply chars in string from mcu to pc*/
#define X_LSB     3
#define Y_MSB     4
#define Y_LSB     5
#define Z_MSB     6
#define Z_LSB     7
#define ROLL_MSB  8
#define ROLL_LSB  9
```

```
#define PITCH_MSB 10
#define PITCH_LSB 11
#define YAW_MSB   12
#define YAW_LSB   13

#define THROTTLE_MSB   2  /* index of servo control request chars     */
#define THROTTLE_LSB   3  /* in string from pc to mcu                  */
#define AILERON_MSB    4
#define AILERON_LSB    5
#define ELEVATOR_MSB   6
#define ELEVATOR_LSB   7
#define RUDDER_MSB     8
#define RUDDER_LSB     9
#define COLLECTIVE_MSB 10
#define COLLECTIVE_LSB 11

/* index variables */
unsigned char TXindex;   /* index of char to be TXed in the buffer      */
unsigned char RXindex;   /* number of chars which have been received    */

/* TX Buffers for HC11, RX Buffers for PC                                */
/* position values ack TX buffer                                         */
unsigned char PosAckBuff[POS_ACK_STRING_LENGTH];
/* servo control ack TX buffer                                           */
unsigned char SerAckBuff[SER_ACK_STRING_LENGTH];

/* RX Buffers for HC11, TX Buffers for PC                                */
/* Buffer of position request                                            */
unsigned char PosReqBuff[POS_REQ_STRING_LENGTH];
/* Buffer of received Data                                               */
unsigned char SerReqBuff[SER_REQ_STRING_LENGTH];
unsigned char WorkSCSR;            /* status register of the SCI         */
unsigned char WorkRXdata;          /* received data register             */
unsigned char WorkCommandChar;     /* Command character RXed from PC     */
unsigned char ValidCommandChar;    /* Most recent valid command rx'd     */
unsigned char BadStopChar;         /* char rx'd in place of stop         */

/* Sephamores                                                            */
/* TRUE if there is Noise,Framing error or an Overrun error              */
unsigned char NoiseFraming;
unsigned char Overrun;
/* TRUE if an unknown command char is RXed                               */
unsigned char UnknownCommand1;
unsigned char UnknownCommand2;
unsigned char NoStopChar;
unsigned char CheckSumBad; /* rx'd bad checksum                          */
unsigned char IndexError;  /* Indexing Error                             */
unsigned char RXstream;    /* TRUE implies that a sequence is being RXed */
/* TRUE when a TX should take place                                      */
unsigned char AckWanted = FALSE;
unsigned char OC3triggered;
unsigned char OC4triggered;
unsigned char OC5triggered;

/*-------------------------------------------------------------------*/
/*      Byte operation variables used to concatenate and cut bytes   */
/*-------------------------------------------------------------------*/
unsigned char LSBits;
unsigned char MSBits;           /* LSbits or MSbits to concat or results */
unsigned int  Int_To_Split;
unsigned int  Concat_2B;
```

```
unsigned char PosOrSer = BAD_REQ_COM_CHAR;  /* Temp til chksum and stop  */

/*------------------------------------------------------------------------*/
/*                          VARIABLES DEFINITION                           */
/*------------------------------------------------------------------------*/
/*------------------------------------------------------------------------*/
/*          VARIABLES RELATIVE TO THE INPUT CAPTURE FUNCTIONS               */
/*------------------------------------------------------------------------*/
#define RX_RANK 2             /* Rank of the IC used for Receiver          */
#define RX_NUM (RX_RANK+1)    /* Number of the IC used for Receiver        */
#define APM_CHAN_RANK 2       /* rank of the pulse that gives the APM      */
#define NUM_IC 3              /* number of input captures in the HC11      */
#define NUM_PULSE 8           /* Number of channels to read in a signal    */
                              /* the synchronization pulse is not included*/
#define TCNT_MAX_VAL 0xFFFF   /* Maximum value of the main 16 bit timer    */
#define OVERFLOW_MAX_VAL 0xFF/* Maximum value of the 8 bit overflow        */
                        /* software counter (see type definition)   */
#define SET_PIN TRUE  /* used by to determine next state of output pin    */
#define CLEAR_PIN FALSE /* ditto */

/* 2 mhz, 500 ns per clock cycle of timer, so 2000 clock cycles = 1 ms    */
#define ONE_MS 2000   /* min acceptable time between two RX rising edges */
#define ONE_POINT_TWO_MS 2400
#define ONE_POINT_THREE_MS 2600
#define ONE_POINT_FIVE_MS  3000  /* Servo Midpoint                        */
#define ONE_POINT_SEVEN_MS 3400
#define ONE_POINT_EIGHT_MS 3600
#define TWO_MS 4000   /* max acceptable time between two RX rising edges */
#define SOFTWARE_CORRECTION 0  /* compensates for instruction
                                  execution time before clearing pulse
                                  number of 500 ns ticks to subtract   */
#define INTER_PULSE_DURATION 4000 /* arbitrary delay between individual
                                  servo pulses                          */
#define THROTTLE_LOWER_LIMIT ONE_MS /* should get green light            */
#define THROTTLE_UPPER_LIMIT TWO_MS /* should get red light              */
#define AILERON_LOWER_LIMIT ONE_MS
#define AILERON_UPPER_LIMIT TWO_MS
#define ELEVATOR_LOWER_LIMIT ONE_MS
#define ELEVATOR_UPPER_LIMIT TWO_MS
#define RUDDER_LOWER_LIMIT ONE_MS
#define RUDDER_UPPER_LIMIT TWO_MS
#define COLLECTIVE_LOWER_LIMIT ONE_POINT_TWO_MS /* controls bind          */
#define COLLECTIVE_UPPER_LIMIT ONE_POINT_SEVEN_MS /* controls bind        */

/* default servo values for initialization                                */
#define THROTTLE_DEFAULT   THROTTLE_LOWER_LIMIT
#define ELEVATOR_DEFAULT   ONE_POINT_FIVE_MS
#define AILERON_DEFAULT    ONE_POINT_FIVE_MS
#define RUDDER_DEFAULT     ONE_POINT_FIVE_MS
#define TO_GYRO_DEFAULT    ONE_POINT_FIVE_MS
#define COLLECTIVE_DEFAULT COLLECTIVE_LOWER_LIMIT

#define NUM_PA 8                          /* number of pins in port A */
#define NUM_PB 8                          /* number of pins in port B */
#define NUM_OC 7                 /* number of output compares of HC11 */
#define MAX_SERVO_LABEL 5        /* what is largest number of servo   */

/* labels are for indices into arrays and match servo assignments on rx  */
/* these numbers are also the numbers on the servo output block          */
#define FROM_GYRO  0 /* input from gyro */
```

```
#define THROTTLE   1 /* store throttle control value in PB's array pos 1 *
#define AILERON    2 /* store aileron control value in PA's array pos 2  *
#define ELEVATOR   3 /* store elevator control value in PB's array pos 3 *
#define RUDDER     4 /* store rudder control value in PA's array pos 4   *
#define TO_GYRO    5 /* 1.5 ms pulse always to gyro                      *
#define COLLECTIVE 6 /* store pitch control value in PA's array pos 6    *


/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
/*          6811 Evaluation Board Hardware Definitions                     *
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
#define LATCH_SCI 0x4000  /* address of flipflop to enable RX of data     *

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
/*          Stand hardware definitions                                     *
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
#define LENGTH_ARM1  64      /* length of lower arm in centimeters         *
#define LENGTH_ARM2  94      /* length of elevation arm in centimeters     *
/* pot number match A/D block numbers but do not match Port E pin numbers*
#define AZ_POT          0      /* PE 4 */
#define ROLL_POT        1      /* PE 1 */
#define EL_POT          2      /* PE 5 */
#define YAW_POT         3      /* PE 2 */
#define H_POT           4      /* PE 6 */
#define PITCH_POT       5      /* PE 3 */
#define SHOW_CAL_VALUE 6       /* used by FAMECAL.C to show cal values  */
#define QUIT_VALUE      8      /* used by FAMECAL.C to quit             */
#define EDIT_CAL_VALUE 7       /* used by FAMECAL.C to edit cal values  */

/*~~~~~~~~~~~~~~~~~~~~~~~~~ TYPE DEFINITION ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/* TimeMemo:       type associated to each input capture function IC    */
/*                                                                       */
/* Time[2]:    used to store two consecutive values of the IC register   */
/*                 ie time of capture of 2 consecutive edge              */
/*                                                                       */
/* OverflowCount:  software 8 bit counter to count the number of main   */
/*         timer overflows that has occur between the capture of two    */
/*         consecutive edges by a given input capture pin               */
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
typedef struct
 {
 unsigned int Time[2];
 unsigned char OverflowCount;  /* software overflow counter */
 } TimeMemo;

TimeMemo IC[NUM_IC];/* Counter overflow & array to save time for each IC */
/* Values of the receiver channels belonging to the same frame          */
unsigned int PW_RX[NUM_PULSE];
unsigned int BUFF_RX[NUM_PULSE]; /* Buffer to store the received values  */
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*            VARIABLES RELATIVE TO THE RECEIVER INPUT                   */
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*  PWtempo:     When two successive edge have been detected by the Input */
/*               Capture number (i) the time betweenn them is computed and*/
/*               and stored in PWtempo[i-1]                              */
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
unsigned int PWtempo[NUM_IC];  /* array of PWtempo for each input capture*/
unsigned char PW_Read[NUM_PULSE]; /* PW_Read[i] indicates if channel(i)  */
                                  /* has been already read              */
unsigned char Synch;              /* Set when the system is synchronized */
```

```
                                  /* to the receiver signal               *
unsigned char RXFirstInter; /* Set when capture of rx signal has stopped.*
                            /* Cleared when first edge is detected after *
                            /* capture has been enabled again            *
unsigned char RX_Read;      /* Set to 1 if a full frame has been read    *

/*------------------------------------------------------------------------*
/*          VARIABLES RELATIVE TO THE OUTPUT COMPARE FUNCTIONS            *
/*------------------------------------------------------------------------*
/* Generates signal on a Port A pin & a Port B pin using one output       *
/* compare.                                                               *
/* arrays are larger than necessary so that output block numbers can be   *
/* used as indices.                                                       *
/*------------------------------------------------------------------------*
unsigned int ThighPA[NUM_PA+1];
unsigned int ThighPB[NUM_PB+1]; /* Buffer to store time high for each pin*
unsigned int AIndex;
unsigned int BIndex;
unsigned int  ServoStatus[NUM_OC];  /* remembers which servo is active    *

/*------------------------------------------------------------------------*/
/*                        HC11 REGISTER VARIABLES                         *
/*------------------------------------------------------------------------*/
unsigned int  *TCNT_Add;     /* main timer counter register               */
unsigned char *TMSK2_Add;    /* main timer interrupt mask                 */
unsigned char *TFLG2_Add;    /* maintimer flag register                   */

unsigned int *IC_Add[NUM_IC];  /* pointer to input capture registers.     */
unsigned int *OC_Add[NUM_OC];  /* pointer to output compare registers     */

unsigned char *TMSK1_Add;   /* output compare and input capture int masks*/
unsigned char *TFLG1_Add;   /* output compare and input capture flags     */
unsigned char *TCTL2_Add;
unsigned char *TCTL1_Add;
unsigned char *OC1D_Add;
unsigned char *OC1M_Add;    /* Output 1 control                           */
unsigned char *PACTL_Add;

/*--------------------------- SCI REGISTERS ----------------------------*/
unsigned char *SCSR_Add;        /* status register of the SCI:flags      */
unsigned char *SCDR_Add;        /* received and transmit data register   */
unsigned char *SCCR2_Add;       /* interrupt enables and state of SCI    */
unsigned char *SCCR1_Add;       /* data format 8 or 9 bits               */
unsigned char *BAUD_Add;        /* baud rate register                    */
unsigned char *LATCH_SCI_Add;   /* software controllable latch to connect
                            pin PD0 to I/O connector          */

/*~~~ Port A & B registers: for sending pulses using output compare -----*/
unsigned char *PORTB_Add;
unsigned char *PORTA_Add;

/*-------------------------  Port D registers  -------------------------*/
unsigned char *PORTD_Add;       /* Port D                                */
unsigned char *DDRD_Add;        /* Data Direction for Port D             */
unsigned char *SPCR_Add;        /* SPI Control Register                  */

/*--------------------- EEPROM programming registers ------------------*/
unsigned char *PPROG_Add;       /* HC11 registers                        */
unsigned char *CONFIG_Add;

/*----------------------------------------------------------------------*/
```

```
/*                              A/D CONVERSION                                   *
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
double PiOverTwo;              /* compute the constant for later use    *
double PiOverFour;             /* compute the constant for later use    *
unsigned char *OPTION_Add;     /* HC11 registers                        *
unsigned char *ADCTL_Add;      /* Control Register for A/D converter    *
unsigned char *ADR1_Add;       /* loc where converted values are stored *
unsigned char *ADR2_Add;
unsigned char *ADR3_Add;
unsigned char *ADR4_Add;

/* Bit patterns written to ADCTL to trigger A/D converters                       *
#define PE0to3_ADCTL 0x10     /* Scan=off, Multiple channel,
                                 Convert Port E channels 0 through 3             *
#define PE4to7_ADCTL 0x14     /* Scan=off, Multiple channel,
                                 Convert Port E channels 4 through 7             *

#define PE0_ADCTL 0x00  /* Value to load ADCTL with to measure pin PE0   */
#define PE1_ADCTL 0x01  /* Value to load ADCTL with to measure pin PE1   */
#define PE2_ADCTL 0x02  /* Value to load ADCTL with to measure pin PE2   */
#define PE3_ADCTL 0x03  /* Value to load ADCTL with to measure pin PE3   */
#define PE4_ADCTL 0x04  /* Value to load ADCTL with to measure pin PE4   */
#define PE5_ADCTL 0x05  /* Value to load ADCTL with to measure pin PE5   */
#define PE6_ADCTL 0x06  /* Value to load ADCTL with to measure pin PE6   */
#define PE7_ADCTL 0x07  /* Value to load ADCTL with to measure pin PE7   */


/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*          variables relative to the position determination                    */
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/* Stand potentiometer angles used to determine the Cartesian location          */
double angle1z;
double angle2z;
double angle2x;

/* Potentiometer angles used to determine the rotational location               */
double angle3pitch;
double angle3roll;
double angle3yaw;

/* First Voltage from AD converter                                               */
unsigned char V3pitch0;
unsigned char V3roll0;
unsigned char V3yaw0;
unsigned char V2z0;
unsigned char V2x0;
unsigned char V1z0;

/* Second Voltage from AD converter                                              */
unsigned char V3pitch1;
unsigned char V3roll1;
unsigned char V3yaw1;
unsigned char V2z1;
unsigned char V2x1;
unsigned char V1z1;

#define DLY10  0x4E40      /* delay of 10 ms in term of main timer cycle */

/*  Global variables used to cut Double_To_Split into 4 unsigned char           */
unsigned char Byte0;
unsigned char Byte1;
```

```
unsigned char Byte2;
unsigned char Byte3;
double Double_To_Split;

/* Cartesian position of the Helicopter                                    */
int Xcord;
int Ycord;
int Zcord;

/* rotational position of the Helicopter                                   */
int Roll;
int Pitch;
int Yaw;

/* constants relative to coordinate conversion and scaling for tx          */
double RadiansToDegrees;
double AngleTxScaleFactor;
double PositionTxScaleFactor;

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*       variables for retrieving calibration values from the EEPROM       */
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/* EEPROM Slope values                                                     */
double *Slope1z_Add;
double *Slope2z_Add;
double *Slope2x_Add;
double *Slope3pitch_Add;
double *Slope3roll_Add;
double *Slope3yaw_Add;

/* EEPROM DC offset voltage values                                         */
unsigned char *V1z0_Add;
unsigned char *V2x0_Add;
unsigned char *V2z0_Add;
unsigned char *V3pitch0_Add;
unsigned char *V3roll0_Add;
unsigned char *V3yaw0_Add;

/* EEPROM addresses of calibration values for potentiometers               */
#define SLOPE1z_ADDRESS      0x7f11
#define V1z_ADDRESS          0x7f15

#define SLOPE2x_ADDRESS      0x7f21
#define V2x_ADDRESS          0x7f25

#define SLOPE2z_ADDRESS      0x7f31
#define V2z_ADDRESS          0x7f35

#define SLOPE3pitch_ADDRESS  0x7f41
#define V3pitch_ADDRESS      0x7f45

#define SLOPE3roll_ADDRESS   0x7f51
#define V3roll_ADDRESS       0x7f55

#define SLOPE3yaw_ADDRESS    0x7f61
#define V3yaw_ADDRESS        0x7f65
```

```
/*-------------------------------------------------------------------------*
/*  Declaration of the H11 register addresses defined in the library       *
/*  c:\introl\kjh\kjhstart.o11                                             *
/*-------------------------------------------------------------------------*
extern unsigned char H11PORTA;        /* i/o port A                       *
extern unsigned char H11PIOC;         /* parallel i/o control register   *
extern unsigned char H11PORTC;        /* i/o port C                       *
extern unsigned char H11PORTB;        /* i/o port B                       *
extern unsigned char H11PORTCL;       /* alternate latch port C           *
extern unsigned char H11DDRC;         /* data direction for port C        *
extern unsigned char H11PORTD;        /* i/o port D                       *
extern unsigned char H11DDRD;         /* i/o data direction for port D    */
extern unsigned char H11PORTE;        /* i/o port D                       */
extern unsigned char H11CFORC;        /* compare force register           */
extern unsigned char H11OC1M;         /* OC1 action mask register         */
extern unsigned char H11OC1D;         /* OC1 action data register         */

extern unsigned int H11TCNT;          /* timer counter register           */
extern unsigned int H11TIC1;          /* input capture register 1         */
extern unsigned int H11TIC2;          /* input capture register 2         */
extern unsigned int H11TIC3;          /* input capture register 3         */
extern unsigned int H11TOC1;          /* output compare register 1        */
extern unsigned int H11TOC2;          /* output compare register 2        *
extern unsigned int H11TOC3;          /* output compare register 3        *
extern unsigned int H11TOC4;          /* output compare register 4        */
extern unsigned int H11TOC5;          /* output compare register 5        */

extern unsigned char H11TCTL1;        /* timer control register 1         */
extern unsigned char H11TCTL2;        /* timer control register 2         */
extern unsigned char H11TMSK1;        /* main timer interrupt mask 1      */
extern unsigned char H11TFLG1;        /* main timer interrupt flag 1      */
extern unsigned char H11TMSK2;        /* main timer interrupt mask 2      */
extern unsigned char H11TFLG2;        /* misc timer interrupt flag 2      */
extern unsigned char H11PACTL;        /* pulse acc control register       */
extern unsigned char H11PACNT;        /* pulse acc count register         */
extern unsigned char H11SPCR;         /* SPI control register             */
extern unsigned char H11SPSR;         /* SPI status register              */
extern unsigned char H11SPDR;         /* SPI data in/out                  */
extern unsigned char H11BAUD;         /* SCI baud rate control            */
extern unsigned char H11SCCR1;        /* SCI control register 1           */
extern unsigned char H11SCCR2;        /* SCI control register 2           */
extern unsigned char H11SCSR;         /* SCI status register              */
extern unsigned char H11SCDR;         /* SCI data                         */
extern unsigned char H11ADCTL;        /* A to D control register          */
extern unsigned char H11ADR1;         /* A to D result 1                  */
extern unsigned char H11ADR2;         /* A to D result 2                  */
extern unsigned char H11ADR3;         /* A to D result 3                  */
extern unsigned char H11ADR4;         /* A to D result 4                  */
extern unsigned char H11OPTION;       /* System configuration options     */
extern unsigned char H11COPRST;       /* arm /reset COPtimer circutry     */
extern unsigned char H11PPROG;        /* EEPROM programming control       */
extern unsigned char H11HPRIO;        /* highest priority I bit and misc*/
```

```
extern unsigned char H11INIT;          /* RAM /io mapping register          *
extern unsigned char H11TEST1;         /* factory test control               *
                              /* COP, ROM, &EEPROM enables      *
extern unsigned char H11CONFIG;




/**********************************************************************
/*                              PROTOTYPES                               *
/**********************************************************************
/* functions for measuring pots and calculating positions and angles    *
void InitADconverter(void);
void MeasureAngles(void);
void PositionDetermination(void);

/* functions used for both TX & RX                                       *
void SCI_ISR(void);  /* ISR for transmission or reception over SCI      *
unsigned char Checksum(int maxnum, unsigned char CheckArray[]);

/* Transmission / Acknowlegement prototypes                              */
void SCI_init_TX(void);    /* initializes TX to PC                       */
void SCI_TX_ISR(void);     /* ISR that transmits position values to PC   */
void TXfirst(void);
void FillPosAckBuff(void);
void FillSerAckBuff(void);
void Split_Int(int i);   /* function that calls asm function below        */
void split_int(void);    /* assy lang to prepare pos values for TX to PC */

/* Reception of Position value requests & servo control requests         */
void SCI_init_RX(void);  /* initialize for reception over SCI            *
void SCI_RX_ISR(void);   /* ISR that receives and stores characters      *
void ReinitializeReceive(void);
void SaveReg(void); /* reads RX buffers & saves values to work registers */
void RXerror(void);
void CharRX(void);
void CompleteRXstream(void);  /* verify that checksum is valid & store
                                 servo control data in array             */
/* function to call asm routine below                                    */
int  Concat_Int(unsigned char MSbits,unsigned char LSbits);
/* asm routine for concatenating servo control values                    */
void concat_int(void);
void ClearWorkVar(void);

/* Servo Control Prototypes                                              */
void InitOC(int OCnum, int Enable);
void DecodeAndStoreServoString(void);
void InitServos(void);
void SendPulsePA_PB(int OCnum, int PAnum, int PBnum);
void OC3_ISR(void);
void OC4_ISR(void);
void OC5_ISR(void);

void InitOverflow(void);     /* enables ISR below                         */
void TCNT_Overflow(void);    /* ISR counts number counter overflows       */

/* functions for doing basic bit operations on register settings         */
void InitializeConstantVariables(void);
void InitializeVectorTable(void);
```

```
void InitPointer(void);
void ClearBit(unsigned char *pointer,int NumBit);
void ClearFlag(unsigned char *pointer,int NumBit);
void SetBit(unsigned char *pointer,int NumBit);
unsigned char GetBit(unsigned char *pointer,int NumBit);
unsigned char GetBit_Char(unsigned char reg,int NumBit);
```

```
/****************************************************************************
/*                         George Mason University                          *
/*         Department of Electrical and Computer Engineering               *
/*                                                                         *
/* File Name:  FAMEmain.C                                                  *
/*                                                                         *
/*  Authors: Bertina Ho-Mock-Qai, Darrell Duane,  Ken Hintz                *
/*  Update History:  Version 1.0, February 23, 1992                        *
/*                                                                         *
/* *** indicates functions and their prototypes                            *
/* --- indicates ISRs                                                      *
/*                                                                         *
/****************************************************************************
#include "c:\introl\kjh\KJHSTDIO.h"
#include "c:\introl\include\MATH.h"
#include "c:\introl\dd8\FAMEDEF.H"
#include "c:\introl\dd8\FAMELIB.c"
#include "c:\introl\dd8\FAMEINIT.c"
#include "c:\introl\dd8\FAMEISR.c"

/****************************************************************************
/* resets receive indices and initializes rx interrupts                    *
/****************************************************************************
void ReinitializeReceive(void)

{
 RXindex      = 0;              /* reset index */
 RXstream     = FALSE;
 EnableRXint();                 /* Enable RX interrupts */
 CheckSumBad  = FALSE;
}


/****************************************************************************/
/*      Function to initialize the different reception variables           */
/****************************************************************************/
void ClearWorkVar(void)

{
int i;

 WorkCommandChar=0;
/* clear servo control request buffer                                      */
 for(i = 0; i < SER_REQ_STRING_LENGTH; i++)
  {
  SerReqBuff[i]=0;
  }
/* clear entire position value request buffer                              */
 for(i = 0; i < POS_REQ_STRING_LENGTH; i++)
  {
  PosReqBuff[i]=0;
  }
}
```

```
/*****************************************************************************
/*  Function called when a complete servo control sequence has been rx'd   *
/*  verifies checksum correct, if yes it calls StoreData, then the          *
/*  servo controls are output to the Output Compare pins                    *
/*  only called when servo control request values RXed from PC              *
/*****************************************************************************
 oid DecodeAndStoreServoString(void)

{
unsigned int LocalUI;

 DisableRXint();
 LocalUI = Concat_Int(SerReqBuff[THROTTLE_MSB],
                      SerReqBuff[THROTTLE_LSB]);
 if (   (LocalUI >= THROTTLE_LOWER_LIMIT)
     &&(LocalUI <= THROTTLE_UPPER_LIMIT))
  {
  ThighPB[THROTTLE] = LocalUI;
  }
 LocalUI = Concat_Int(SerReqBuff[AILERON_MSB],
                      SerReqBuff[AILERON_LSB]);
 if (   (LocalUI >= AILERON_LOWER_LIMIT)
     &&(LocalUI <= AILERON_UPPER_LIMIT))
  {
  ThighPA[AILERON] = LocalUI;
  }
 LocalUI = Concat_Int(SerReqBuff[ELEVATOR_MSB],
                      SerReqBuff[ELEVATOR_LSB]);
 if (   (LocalUI >= ELEVATOR_LOWER_LIMIT)
     &&(LocalUI <= ELEVATOR_UPPER_LIMIT))
  {
  ThighPB[ELEVATOR] = LocalUI;
  }
 LocalUI = Concat_Int(SerReqBuff[RUDDER_MSB],
                      SerReqBuff[RUDDER_LSB]);
 if (   (LocalUI >= RUDDER_LOWER_LIMIT)
     &&(LocalUI <= RUDDER_UPPER_LIMIT))
  {
  ThighPA[RUDDER] = LocalUI;
  }
 LocalUI = Concat_Int(SerReqBuff[COLLECTIVE_MSB],
                      SerReqBuff[COLLECTIVE_LSB]);
 if (   (LocalUI >= COLLECTIVE_LOWER_LIMIT)
     &&(LocalUI <= COLLECTIVE_UPPER_LIMIT))
  {
  ThighPA[COLLECTIVE] = LocalUI;

 ReinitializeReceive();
} /* end function Complete RX stream */
```

```
/*************************************************************************
/* Function called after a complete position request sequence is received*
/* It triggers the A/D converters to sample the potentiometers angles,    *
/* and fills the TX buffer with the measured values.                      *
/*************************************************************************
void FillPosAckBuff(void)

{
 MeasureAngles();      /* trigger A/D conv., & measure all angles */
 PositionDetermination();
 PosAckBuff[START_CHAR_INDEX]            = START_CHAR;
 PosAckBuff[COM_CHAR_INDEX]              = POS_ACK_COM_CHAR;
 PosAckBuff[POS_ACK_STRING_LENGTH - 1] = STOP_CHAR;

 Split_Int(Xcord);  /* send X cartesian coordinate of helicopter to PC */
 PosAckBuff[X_MSB] = MSBits;
 PosAckBuff[X_LSB] = LSBits;

 Split_Int(Ycord);  /* send Y cartesian coordinate of helicopter to PC */
 PosAckBuff[Y_MSB] = MSBits;
 PosAckBuff[Y_LSB] = LSBits;

 Split_Int(Zcord);  /* send Z cartesian coordinate of helicopter to PC */
 PosAckBuff[Z_MSB] = MSBits;
 PosAckBuff[Z_LSB] = LSBits;

 Split_Int(Roll);    /* send roll to PC                                    */
 PosAckBuff[ROLL_MSB] = MSBits;
 PosAckBuff[ROLL_LSB] = LSBits;

 Split_Int(Pitch);  /* send pitch to PC                                    */
 PosAckBuff[PITCH_MSB] = MSBits;
 PosAckBuff[PITCH_LSB] = LSBits;

 Split_Int(Yaw);    /* Send yaw to PC                                      */
 PosAckBuff[YAW_MSB] = MSBits;
 PosAckBuff[YAW_LSB] = LSBits;

/* Calculate checksum */
 PosAckBuff[POS_ACK_STRING_LENGTH - 2]
     = Checksum(POS_ACK_STRING_LENGTH, PosAckBuff);
}


/*************************************************************************/
/*  Calculates X,Y,Z positional & rotational coordinates using the       */
/*  values read from the A/D converters which measure the potentiometers.*/
/*  double precision numbers from positions are scaled before converting */
/*  to integer before transmission.  The position scale factor can be fou*/
/*  in FAMEDEF.h as POSITION_TX_SCALE_FACTOR.                            */
/*  The same goes for angles except they are converted to degrees also an*/
/*  their scaling factor is ANGLE_TX_SCALE_FACTOR.                       */
/*************************************************************************/
void PositionDetermination(void)

{
double d;

 d = (double)LENGTH_ARM1 * cos(angle1z)
     + (double)LENGTH_ARM2 * cos(angle1z + angle2z) * cos(angle2x);
 Xcord = (int)(PositionTxScaleFactor * d);
```

```
 d = (double)LENGTH_ARM1 * sin(angle1z)
     + (double)LENGTH_ARM2 * (sin(angle1z + angle2z)) * cos(angle2x);
 Ycord = (int) (PositionTxScaleFactor * d);
 d = (double)LENGTH_ARM2 * sin(angle2x);
 Zcord = (int)( PositionTxScaleFactor * d);
 Pitch = (int)( RadiansToDegrees * AngleTxScaleFactor * angle3pitch);
 Roll = (int)( RadiansToDegrees * AngleTxScaleFactor * angle3roll);
 Yaw = (int)(  RadiansToDegrees * (angle3yaw + angle1z + angle2z)
            * AngleTxScaleFactor );
}  /* end function PositionDetermination */


/***************************************************************************
/* Function that calculates the angles measured by the potentiometers     *
/* Measure values of A/D converters for cartesian location calc.          *
/***************************************************************************
void MeasureAngles(void)

{
/* convert voltages measured on pins PE4 through PE7                       *
 *ADCTL_Add = PE4to7_ADCTL;
 while(GetBit(ADCTL_Add,7)==0);   /* wait for conversion to complete */
/* Calibration of the cartesian angle values using the EEPROM coefficients *
 angle1z = ( (double)*Slope1z_Add *
             (double)((int)*ADR3_Add - (int)*V1z0_Add) );
 angle2x = ( (double)*Slope2x_Add *
             (double)((int)*ADR2_Add - (int)*V2x0_Add) );
 angle2z = ( (double)*Slope2z_Add *
             (double)((int)*ADR1_Add - (int)*V2z0_Add) );

*ADCTL_Add=PE0to3_ADCTL;          /* convert voltages measured on pins PE0
through PE3  */
 while(GetBit(ADCTL_Add,7)==0);  /* wait for conversion to complete */
/* Calibration of the rotational angle values using the EEPROM coefficients */
 angle3pitch = ( (double)*Slope3pitch_Add * (double)((int)*ADR4_Add
               - (int)*V3pitch0_Add) );
 angle3roll  = ( (double)*Slope3roll_Add * (double)((int)*ADR2_Add
               - (int)*V3roll0_Add) );
 angle3yaw   = ( (double)*Slope3yaw_Add * (double)((int)*ADR3_Add
               - (int)*V3yaw0_Add) );
}  /* end of function ConvertAD */


/***************************************************************************/
/* TXes the first char of the TX buffer, which spawns the TX of the rest */
/* of the buffer.                                                          */
/***************************************************************************/
void TXfirst(void)

{
 TXindex = 0;
 *SCSR_Add = *SCSR_Add; /* TDRE flag cleared after function writes to TDR */
 switch(ValidCommandChar)
  {
  case POS_REQ_COM_CHAR:
          *SCDR_Add = PosAckBuff[TXindex];
                 /* Write to the TDR & clear TDRE flag */
            EnableTXbuffEmptyInt(); /* enable transmit complete interrupt
macro       */
          break;
  case SER_REQ_COM_CHAR:
```

```
            *SCDR_Add = SerAckBuff[TXindex];
                  /* Write to the TDR & clear TDRE flag */
              EnableTXbuffEmptyInt(); /* enable transmit complete interrupt
macro       */
            break;
  default:
            printf("Attempt to TX unknown type in TXfirst() \r");
            break;
  } /* end switch() */
   /* The next value that should be sent by the ISR will have an index = 1 */
} /* end TXfirst */


/**************************************************************************
/*  Main function to receive a character according to the control word    */
/*  the corresponding functions are called, 2 possible sequences can be   */
/*  received  "position sequence" and the "servo control sequence"        */
/**************************************************************************/
void CharRX(void)

{
 if(RXindex == START_CHAR_INDEX)  /* Determine command, start only rx'd */
  {
  RXindex++;
  WorkCommandChar = WorkRXdata;   /* WCC used for further character routing*/
  switch(WorkCommandChar)
   {
   case POS_REQ_COM_CHAR:  /* is this a position request? */
     PosReqBuff[RXindex] = WorkRXdata;  /* Store command char */
     break;
   case SER_REQ_COM_CHAR:  /* is this servo control request? */
     SerReqBuff[RXindex] = WorkRXdata;  /* Store command char for Checksum */
     break;
   default:             /* is this an unknown request? */
     UnknownCommand1 = TRUE;
     RXindex = 0;
     RXstream = FALSE;
     break;
   } /* end switch() for command character */
  }  /* end if this is the command character */
 else   /* any character other than the start char or command char */
  {
  RXindex++;
  switch(WorkCommandChar)
   {
   case POS_REQ_COM_CHAR:  /* is this a position request (no data)? */
    if(  (RXindex    == (COM_CHAR_INDEX+1))
      && (WorkRXdata == STOP_CHAR)) /* is this the stop char? */
     {
     ValidCommandChar = WorkCommandChar;
     AckWanted=TRUE;       /* set sephamore to trigger acknowledgement to PC */
     }           /* disable rx inter */
    else  /* error */
     printf("No stop character in position request \r");
     ReinitializeReceive();
     break;
   case SER_REQ_COM_CHAR:  /* is this servo control sequence (need data)? */
    if( (RXindex > COM_CHAR_INDEX)
      &&(RXindex < (SER_REQ_STRING_LENGTH - 1))) /* All but stop char*/
     {
      SerReqBuff[RXindex] = WorkRXdata;
```

```
    } /* save servo control or checksum values to this array */
   else if(RXindex == SER_REQ_STRING_LENGTH - 1) /* RXINDEX -> STOP CHAR */
     {
     if(WorkRXdata == STOP_CHAR) /* is it the stop char? */
      {
      SerReqBuff[RXindex] = WorkRXdata;
      if(SerReqBuff[SER_REQ_STRING_LENGTH - 2]
         == Checksum(SER_REQ_STRING_LENGTH, SerReqBuff))
       {                                      /* and checksum OK    */
       ValidCommandChar = WorkCommandChar;
       AckWanted = TRUE;           /* Let foreground know new command */
       }
      else
       {
       CheckSumBad = TRUE;
       ReinitializeReceive();
       }
      NoStopChar = FALSE;
      }  /* disable RX while TXing ack */
     else /* error */
      {
      NoStopChar=TRUE;
      BadStopChar = WorkRXdata;
      ReinitializeReceive();
      }
     }
    /*else*/ /* indexing error */
    /* {
     IndexError=TRUE;
     NoStopChar = FALSE;
     ReinitializeReceive();
     }*/
   break;

  default:              /* is this an unknown request? */
   UnknownCommand2 = TRUE;
   ReinitializeReceive();
   break;
  } /* end switch() */
 } /* end else this is not the Command Char */
} /* end of function CharRX */


/**************************************************************************/
/* Main 68EVB FAME program                                                */
/**************************************************************************/
int main()

{
 asm("disint SEI\n");                    /* disable global interrupt of HC11 */
 InitializeConstantVariables();
 InitializeVectorTable();                /* init ISR jump table in RAM       */
 InitPointer();        /* Initialize values of pointers to HC11 registers */
 InitServos();         /* initialize durations for high & low values
                          output compares (servo control)                  */
 InitADconverter();    /* initialize OPTION register for A/D converters    */
 ClearWorkVar();       /* Initialize the communication buffers             */
 InitOC(OC3_NUM, TRUE); /* initialize Output Compare #3 set time=0,
                           clear flag, enable interrupt                    */
 InitOC(OC4_NUM, FALSE);/* ditto for #4, no enable                         */
 InitOC(OC5_NUM, FALSE);/* ditto for #5, no enable                         */
```

```
SCI_init_TX();        /* Initialize the SCI transmitter
SCI_init_RX();        /* initialize the SCI receiver
asm("inter CLI\n");  /* enable global interrupt of HC11
while(TRUE)          /* endless loop awaiting ISRs & semaphores */
 {
 if(Overrun)
  {
  Overrun=FALSE;
  printf("Overrun\r");
  }
 if(NoiseFraming)
  {
  NoiseFraming=FALSE;
  printf("Noise/Framing\r");
  }
 if(UnknownCommand1)
  {
  UnknownCommand1=FALSE;
  printf("Unknown Command 1 Char RXed from PC\r");
  }
 if(UnknownCommand2)
  {
  UnknownCommand2=FALSE;
  printf("Unknown Command 2 Char RXed from PC\r");
  }
 if(NoStopChar)
  {
  NoStopChar=FALSE;
  printf("Character RXed where stop char expected not a stop char.\r");
  printf("Character RXed instead = %c \r", BadStopChar);
  }
 if(IndexError)
  {
  IndexError=FALSE;
  printf("Index Error\r");
  }
 if(OC3triggered)
  {
  OC3triggered=FALSE;
  }
 if(OC4triggered)
  {
  OC4triggered=FALSE;
  }
 if(OC5triggered)
  {
  OC5triggered=FALSE;
  }
 if(AckWanted)
  {
  putchar('A');  /*(int)0x41);      A   */
  putchar('!');  /*(int)0x21);      !   */
  AckWanted = FALSE;  /* Reset Semaphore */
  switch(ValidCommandChar)
   {
   case POS_REQ_COM_CHAR:     /* Set Sephamore to TX Position Values Request
Acknowledgement */
     FillPosAckBuff();  /* Fill the TX buffer with the position values */
     TXfirst();
     ReinitializeReceive();
     break;
```

```
    case SER_REQ_COM_CHAR:      /* Set Sephamore to TX Servo Control Request
Acknowledgement */
     DecodeAndStoreServoString();
     FillSerAckBuff();  /* Fill the TX buffer with the position values *
     TXfirst();
     ReinitializeReceive();
     break;
    default:
     printf("Invalid Acknowlegement type requested.\r");
     ReinitializeReceive();
     break;
    } /* end switch */
   } /* end if AckWanted */
  }  /* while true */
} /* main routine */
```

```
/*************************************************************************.
/*                         George Mason University                       .
/*          Department of Electrical and Computer Engineering            .
/*                                                                       .
/* File Name:  FAMEINIT.C                                                .
/*                                                                       .
/*  Authors: Bertina Ho-Mock-Qai, Darrell Duane,  Ken Hintz              .
/*  Update History:  Version 1.0, February 23, 1992                      .
/*                                                                       .
/*                                                                       .
/**************************************************************************.
/* Initializes some constants so that they do not have to be computed     .
/* repeatedly during program execution                                    .
/**************************************************************************.
void InitializeConstantVariables(void)

{
 PiOverTwo  = PI / 2.0;
 PiOverFour = PI / 4.0;
 RadiansToDegrees = RADIANS_TO_DEGREES;
 AngleTxScaleFactor = ANGLE_TX_SCALE_FACTOR;  /* scaled integers are sent*/
 PositionTxScaleFactor = POSITION_TX_SCALE_FACTOR;
}


/***************************************************************************/
/* Not Complete, add vectors as needed                                     */
/***************************************************************************/
void InitializeVectorTable(void)

{
unsigned char *TempPointer;

/*  Vector to SCI ISR */
 TempPointer         = (unsigned char *)VSCI;
 *TempPointer        = (unsigned char) JUMPEXTENDED;
 *(TempPointer + 1) =
            (unsigned char) ((0xff00 & ((unsigned int)(&SCI_ISR))) >> 8);
 *(TempPointer + 2) =
            (unsigned char) (0x00ff & ((unsigned int)(&SCI_ISR)));


/*  Vector to OC3 ISR */
 TempPointer         = (unsigned char *)VTOC3;
 *TempPointer        = (unsigned char) JUMPEXTENDED;
 *(TempPointer + 1) =
                (unsigned char) ((0xff00 & ((unsigned int)(&OC3_ISR))) >> 8);
 *(TempPointer + 2) =
                (unsigned char) (0x00ff & (unsigned int) (&OC3_ISR) );

/*  Vector to OC4 ISR */
 TempPointer         = (unsigned char *)VTOC4;
 *TempPointer        = (unsigned char) JUMPEXTENDED;
 *(TempPointer + 1) =
                (unsigned char) ((0xff00 & ((unsigned int)(&OC4_ISR))) >> 8);
 *(TempPointer + 2) =
                (unsigned char) (0x00ff & (unsigned int) (&OC4_ISR) );

/*  Vector to OC5 ISR */
 TempPointer         = (unsigned char *)VTOC5;
 *TempPointer        = (unsigned char) JUMPEXTENDED;
```

```
 *(TempPointer + 1) =
                (unsigned char) ((0xff00 & ((unsigned int)(&OC5_ISR)) >> 8);
 *(TempPointer + 2) =
                (unsigned char) (0x00ff & (unsigned int) (&OC5_ISR) );
}


/*************************************************************************
/*    Function assumes that the OC that we used is not OC1               *
/*    (OC1 does not work as the other OC cf HC11 manual)                 *
/*    Number passed is the output compare number less one                *
/*************************************************************************
void InitOC(int OCnum, int Enable)

{
 *OC_Add[OCnum] = 0x0000;      /* set output compare timer indexes to zero *
 /* set register so that nothing is done to A's pin upon next interrupt *
 ClearBit (TCTL1_Add, (9 - ( 2 * OCnum ) ) );
 ClearBit (TCTL1_Add, (8 - (2 * OCnum ) ) );
 ClearFlag(TFLG1_Add,(8- (OCnum + 1)));  /* clear interrupt flag for OCx *
 ServoStatus[OCnum] = B_LOW;
if(Enable)
 {
 SetBit(TMSK1_Add,(8- (OCnum + 1)));     /* enable interrupt for OCx */
 }
}


/*************************************************************************/
/*      Initialize the HC11 flags, baud rate...to enable the reception   */
/*      must be called once before using the SCI transmission features   */
/*************************************************************************/
void SCI_init_TX(void)

{
 SetBit(SCCR2_Add,3);         /* Enable transmission TE=1 */
 DisableTXbuffEmptyInt();     /* sable TX data buffer empty interrupt */
 DisableTXcompleteInt();      /* Disable TX complete interrupt */
}


/*************************************************************************/
/*      Initialize the HC11 flags, baud rate...to enable the reception   */
/*      must be called once before using the SCI reception features      */
/*************************************************************************/
void SCI_init_RX(void)

{
 *LATCH_SCI_Add  = 0x01;   /* enable SCI on HC11 board by setting flip flop*/
 AckWanted       = FALSE;
 Overrun         = FALSE;
 NoiseFraming    = FALSE;
 UnknownCommand1 = FALSE;
 UnknownCommand2 = FALSE;
 NoStopChar      = FALSE;
 IndexError      = FALSE;
 RXstream        = FALSE;

/* baud=4800 assuming 8 MHz clock  */
 SetBit(BAUD_Add,4);
 SetBit(BAUD_Add,5);
```

```
 SetBit(BAUD_Add,0);
 ClearBit(BAUD_Add,1);
 ClearBit(BAUD_Add,2);

 ClearBit(SCCR1_Add,4);       /* format of the datas is 8 bits long */
 ClearBit(SCCR1_Add,3);

 SetBit(SCCR2_Add,2);        /* Enable RX RE=1 */
 ClearBit(SCCR2_Add,4);      /* Disable Idle Line Interrupt Enable */

 ReinitializeReceive();
}  /* end function SCI_init_RX */


/****************************************************************************
/* Initialize the time low for all Output Compares (set to 1ms minimum)   */
/* Initialize the time high for them as well */
/****************************************************************************
void InitServos(void)

{
int i;

 for(i = 0; i <= NUM_PA; i++)
  {
  ThighPA[i]         = (unsigned int)ONE_MS;
  }  /* Set Sephamore to indicate that Port A is low */
 for(i = 0; i <= NUM_PB; i++)
  {
  ThighPB[i]         = (unsigned int)ONE_MS;
  }  /* set Sephamore to indicate that Port B is low */

/* set specific values for particular servo controls */
  ThighPB [THROTTLE]     = (unsigned int)THROTTLE_DEFAULT;
  ThighPA [AILERON]      = (unsigned int)AILERON_DEFAULT;
  ThighPB [ELEVATOR]     = (unsigned int)ELEVATOR_DEFAULT;
  ThighPA [RUDDER]       = (unsigned int)RUDDER_DEFAULT;
  ThighPB [TO_GYRO]      = (unsigned int)TO_GYRO_DEFAULT;
  ThighPA [COLLECTIVE]   = (unsigned int)COLLECTIVE_DEFAULT;
 *TCTL1_Add = 0x00; /* OC2,3,4,5 set register OC disconnected from outputs */
 *TFLG1_Add = 0;                  /* clear all pending IC and OC interrupts */
 *PORTA_Add = 0;                           /* set all servo outputs to zero */
 *PORTB_Add = 0;
 *OC1M_Add  =  0x00; /* disable OC1 (default) */
} /* end initialize high & low times for servo control */


/*****************************************************************************/
/*  Initialization of the HC11 registers to use the ADconverter          */
/*****************************************************************************/
void InitADconverter(void)

{
 SetBit(OPTION_Add,7);      /* ADPDU=1 */
 ClearBit(OPTION_Add,6);    /* CSEL=0  */
 SetBit(OPTION_Add,4);      /* DLY=1   */
}
```

```
/*************************************************************************
/* This routine makes the pointer variables representing the HC11 registers
/*  point to the corresponding addresses                                   *
/*************************************************************************
void InitPointer(void)

{
 TCNT_Add  = &H11TCNT;     /* Main Timer count register */
 TMSK2_Add = &H11TMSK2;  /* Interrupt Mask for timer operations */
 TFLG2_Add = &H11TFLG2;  /* Flag Register for timer operations */

 IC_Add[IC1_NUM] = &H11TIC1;   /* Array of the 3 Input Capture registers *
 IC_Add[IC2_NUM] = &H11TIC2;
 IC_Add[IC3_NUM] = &H11TIC3;

 TMSK1_Add   = &H11TMSK1;  /* IC & OC mask register */
 TFLG1_Add   = &H11TFLG1;  /* IC & OC flag register */
 TCTL2_Add   = &H11TCTL2;  /* ?? */

 OC_Add[OC1_NUM]  = &H11TOC1;   /* Array of the 5 Output Compare registers */
 OC_Add[OC2_NUM]  = &H11TOC2;
 OC_Add[OC3_NUM]  = &H11TOC3;
 OC_Add[OC4_NUM]  = &H11TOC4;
 OC_Add[OC5_NUM]  = &H11TOC5;

 TCTL1_Add   = &H11TCTL1;  /* Timer Control Register #1: for OC5 - OC2 */
 OC1D_Add    = &H11OC1D;    /* Data for OC1 */
 OC1M_Add    = &H11OC1M;    /* Mask for OC1 */
 PACTL_Add   = &H11PACTL;  /* Pulse Accumulator Control register */

 SCSR_Add    = &H11SCSR;    /* SCI registers */
 SCDR_Add    = &H11SCDR;
 SCCR2_Add   = &H11SCCR2;
 SCCR1_Add   = &H11SCCR1;
 BAUD_Add    = &H11BAUD;
 LATCH_SCI_Add = (unsigned char *)LATCH_SCI;   /* Latch to enable the SCI RX
*/
       /* This latch is software controllable and must be initialized */
 PORTB_Add   = &H11PORTB;   /* communication ports */
 PORTA_Add   = &H11PORTA;
 OPTION_Add = &H11OPTION;  /* AD Converter control registers */
 ADCTL_Add   = &H11ADCTL;
 ADR1_Add    = &H11ADR1;   /* AD converter reg that hold the converted values
*/
 ADR2_Add    = &H11ADR2;
 ADR3_Add    = &H11ADR3;
 ADR4_Add    = &H11ADR4;
   /* Calibrated values of the potentiameters programmed in the EEPROM */
 Slope1z_Add = (double *)SLOPE1z_ADDRESS;
 Slope2z_Add = (double *)SLOPE2z_ADDRESS;
 Slope2x_Add = (double *)SLOPE2x_ADDRESS;
 V1z0_Add = (unsigned char *)V1z_ADDRESS;
 V2z0_Add = (unsigned char *)V2z_ADDRESS;
 V2x0_Add = (unsigned char *)V2x_ADDRESS;
 Slope3pitch_Add = (double *)SLOPE3pitch_ADDRESS;
 Slope3roll_Add  = (double *)SLOPE3roll_ADDRESS;
 Slope3yaw_Add   = (double *)SLOPE3yaw_ADDRESS;
 V3pitch0_Add = (unsigned char *)V3pitch_ADDRESS;
 V3roll0_Add  = (unsigned char *)V3roll_ADDRESS;
```

```
 V3yaw0_Add    = (unsigned char *)V3yaw_ADDRESS;
}
```

```
/*-----------------------------------------------------------------------*
/*                       George Mason University                          *
/*           Department of Electrical and Computer Engineering           *
/* File Name: FAMEISR.c                                                  *
/*                                                                       *
/*  Authors: Bertina Ho-Mock-Qai, Darrell Duane,  Ken Hintz              *
/*  Update History:  Version 1.0, February 23, 1992                      *
/*                                                                       *
/*-----------------------------------------------------------------------*

/*-----------------------------------------------------------------------*
/* ISR for OC3, OC4, OC5                                                 *
/* ISR triggered when value in OC3's register = timer value              *
/*-----------------------------------------------------------------------*
void OC3_ISR(void)

{
 switch (ServoStatus[OC3_NUM])
  {
  case B_LOW:
   {
   ServoStatus[OC3_NUM] = A_HIGH;
   /* set output compare to new value                                    *
   *OC_Add[OC3_NUM] = *TCNT_Add + ThighPA[COLLECTIVE]
                   - SOFTWARE_CORRECTION;
   SetBit(PORTA_Add,PA5);        /* set Port A Bit                       *
   ClearFlag ( TFLG1_Add, (7 - OC3_NUM ) ); /* clear int flag            */
   SetBit(TMSK1_Add,(7 - OC3_NUM ) );     /* enable interrupt for OCx    */
   break;
   }
  case A_HIGH:
   {
   /*  auto clear does not seem to work correctly                        *
   ClearBit(PORTA_Add,PA5);        /* cLEAR Port A Bit                   *
   ServoStatus[OC3_NUM] = A_LOW;
   *OC_Add[OC3_NUM] = *TCNT_Add + INTER_PULSE_DURATION
                   - SOFTWARE_CORRECTION;            ; /* delta pulses   */
   ClearFlag ( TFLG1_Add, (7 - OC3_NUM ) ); /* clear int flag            */
   SetBit(TMSK1_Add,(7 - OC3_NUM ) );      /* enable interrupt for OCx   */
   break;
   }
  case A_LOW:
   {
   ServoStatus[OC3_NUM] = B_HIGH;
   /* set output compare to new value                                    */
   *OC_Add[OC3_NUM] = *TCNT_Add + ThighPB[TO_GYRO] - SOFTWARE_CORRECTION;
   SetBit(PORTB_Add,PB2);        /* set Port B Bit                       */
   ClearFlag ( TFLG1_Add, (7 - OC3_NUM ) ); /* clear int flag            */
   SetBit(TMSK1_Add,(7 - OC3_NUM ) );      /* enable interrupt for OCx     *
   break;
   }
  case B_HIGH:
   {
   ClearBit(PORTB_Add,PB2);        /* set Clear Port B Bit               *
   ClearFlag ( TFLG1_Add, (7 - OC3_NUM ) ); /* clear int flag            */
/*disable OC3 interrupt                                                  *
   ClearBit(TMSK1_Add,(7 - OC3_NUM ) );      /* disable interrupt for OC3 */
   ServoStatus[OC3_NUM] = B_LOW;
/* load OC4 timer for short interpulse duration                          *
   *OC_Add[OC4_NUM] = *TCNT_Add + INTER_PULSE_DURATION
                    SOFTWARE_CORRECTION;
```

```
/* enable int for #4 now so it has a chance without being interrupted     *
   ServoStatus[OC4_NUM] = B_LOW;
   ClearFlag ( TFLG1_Add, (7 - OC4_NUM ) ); /* clear int flag              *
   SetBit(TMSK1_Add,(7 - OC4_NUM ) );     /* enable interrupt for OC4     *
   break;
   }
  }
 asm("      RTI\n"); /* asm command:  Return from Interrupt                 *
}


/*-----------------------------------------------------------------------*
/* ISR triggered when value in OC4's register = timer value              *
/*-----------------------------------------------------------------------*
void OC4_ISR(void)  /* ISR triggered when OC4's register = timer value   *

{
 switch (ServoStatus[OC4_NUM])
  {
  case B_LOW:
   {
   ServoStatus[OC4_NUM] = A_HIGH;
   /* set output compare to new value                                   */
   *OC_Add[OC4_NUM] = *TCNT_Add + ThighPA[RUDDER]
                  - SOFTWARE_CORRECTION;
   SetBit(PORTA_Add,PA4);       /* set Port A Bit                       */
   ClearFlag ( TFLG1_Add, (7 - OC4_NUM ) ); /* clear int flag           */
   SetBit(TMSK1_Add,(7 - OC4_NUM ) );     /* enable interrupt for OCx   */
   break;
   }
  case A_HIGH:
   {
   /*  auto clear does not seem to work correctly                       *
   ClearBit(PORTA_Add,PA4);       /* set Port A Bit                     *
   ServoStatus[OC4_NUM] = A_LOW;
   *OC_Add[OC4_NUM] = *TCNT_Add + INTER_PULSE_DURATION
                  - SOFTWARE_CORRECTION;   /* between pulses            *
   ClearFlag ( TFLG1_Add, (7 - OC4_NUM ) ); /* clear int flag           *
   SetBit(TMSK1_Add,(7 - OC4_NUM ) );     /* enable interrupt for OCx   *
   break;
   }
  case A_LOW:
   {
   ServoStatus[OC4_NUM] = B_HIGH;
   /* set output compare to new value                                   */
   *OC_Add[OC4_NUM] = *TCNT_Add + ThighPB[ELEVATOR]
                  - SOFTWARE_CORRECTION;
   SetBit(PORTB_Add,PB1);       /* set Port A Bit                       */
   ClearFlag ( TFLG1_Add, (7 - OC4_NUM ) ); /* clear int flag           */
   SetBit(TMSK1_Add,(7 - OC4_NUM ) );     /* enable interrupt for OCx   */
   break;
   }
  case B_HIGH:
   {
   ClearBit(PORTB_Add,PB1);     /* set Clear Port B Bit                 */
   ClearFlag ( TFLG1_Add, (7 - OC4_NUM ) ); /* clear int flag           */
/*disable OC4 interrupt                                                 */
   ClearBit(TMSK1_Add,(7 - OC4_NUM ) );     /* disable interrupt for OC4 */
   ServoStatus[OC4_NUM] = B_LOW;
/* load OC5 timer for short interpulse duration                         */
   *OC_Add[OC5_NUM] = *TCNT_Add + INTER_PULSE_DURATION
```

```
                - SOFTWARE_CORRECTION;
/* enable int for #5                                                    */
   ServoStatus[OC5_NUM] = B_LOW;
   ClearFlag ( TFLG1_Add, (7 - OC5_NUM ) ); /* clear int flag           */
   SetBit(TMSK1_Add,(7 - OC5_NUM ) );      /* enable interrupt for OCx  */
   break;
   }
  }
 asm("      RTI\n");  /* asm command:  Return from Interrupt           */
}


/*---------------------------------------------------------------------*/
/* ISR triggered when value in OC5's register = timer value             */
/*---------------------------------------------------------------------*/
void OC5_ISR(void) /* ISR triggered when OC5's register = timer value   */

{
 switch (ServoStatus[OC5_NUM])
  {
  case B_LOW:
   {
   ServoStatus[OC5_NUM] = A_HIGH;
   /* set output compare to new value                                   */
   *OC_Add[OC5_NUM] = *TCNT_Add + ThighPA[AILERON]
                    - SOFTWARE_CORRECTION;
   SetBit(PORTA_Add,PA3);      /* set Port A Bit                        */
   ClearFlag ( TFLG1_Add, (7 - OC5_NUM ) ); /* clear int flag           */
   SetBit(TMSK1_Add,(7 - OC5_NUM ) );      /* enable interrupt for OCx  */
   break;
   }
  case A_HIGH:
   {
   ClearBit(PORTA_Add,PA3);      /* set Port A Bit                      */
   ServoStatus[OC5_NUM] = A_LOW;
   *OC_Add[OC5_NUM] = *TCNT_Add + INTER_PULSE_DURATION
                  - SOFTWARE_CORRECTION;  /* INTER PULSE                */
   ClearFlag ( TFLG1_Add, (7 - OC5_NUM ) ); /* clear int flag           */
   SetBit(TMSK1_Add,(7 - OC5_NUM ) );      /* enable interrupt for OCx  */
   break;
   }
  case A_LOW:
   {
   ServoStatus[OC5_NUM] = B_HIGH;
   /* set output compare to new value                                   */
   *OC_Add[OC5_NUM] = *TCNT_Add + ThighPB[THROTTLE] - SOFTWARE_CORRECTION;
   SetBit(PORTB_Add,PB0);      /* set Port A Bit                        */
   ClearFlag ( TFLG1_Add, (7 - OC5_NUM ) ); /* clear int flag           */
   SetBit(TMSK1_Add,(7 - OC5_NUM ) );      /* enable interrupt for OCx  */
   break;
   }
  case B_HIGH:
   {
   ClearBit(PORTB_Add,PB0);      /* set Clear Port B Bit                */
   ClearFlag ( TFLG1_Add, (7 - OC5_NUM ) ); /* clear int flag           */
/*disable OC5 interrupt                                                 */
   ClearBit(TMSK1_Add,(7 - OC5_NUM ) );      /* disable interrupt for OC5 */
/*   *OC_Add[OC5_NUM] = *TCNT_Add;*/ /* set output compare to new value
*/
   ServoStatus[OC5_NUM] = B_LOW;
/* load OC3 timer so that only one roll over is used before next pulses  */
```

```
    *OC_Add[OC3_NUM] = *TCNT_Add; /* set output compare to new value     */
 /* enable int for #3                                                     */
    ServoStatus[OC3_NUM] = B_LOW;
    ClearFlag ( TFLG1_Add, (7 - OC3_NUM ) ); /* clear int flag           */
    SetBit(TMSK1_Add,(7 - OC3_NUM ) );      /* enable interrupt for OCx  */
    break;
    }
   }
  asm("      RTI\n");  /* asm command:  Return from Interrupt            */
 }
 /*----------------------------------------------------------------------*/
 /* ISR for data received or transmitted via the SCI. As there is only one */
 /* physical interrupt for the SCI, function checks to see which activity */
 /* is occuring according to the flags and calls the corresponding function*/
 /*----------------------------------------------------------------------*/
 void SCI_ISR(void)

 {
  if( (*SCSR_Add & 0x2E) != 0x00) /* was this ISR triggered by a char RXed*/
   SCI_RX_ISR();
  else
   if(  ((*SCSR_Add & 0x80) != 0x00) )
         /* was this ISR triggered by TDRE=1 (the TX data register empty) */
    SCI_TX_ISR(); /* transmit acknowledgement values to PC               */
   else
    {
    printf("SCI ISR triggered for reasons unknown. SCSR= 0x%x SCCR2= 0x%x\r",
         *SCSR_Add, *SCCR2_Add);
    DisableTXbuffEmptyInt();
    }
 asm("      RTI\n");  /* asm command Return From Interrupt               */
 }  /* end SCI_ISR */

 /*----------------------------------------------------------------------*/
 /*           Main ISR to transmit characters to the work station.       */
 /* Transmits second through last characters of string.                  */
 /* Sending of first character done by TXfirst                           */
 /*----------------------------------------------------------------------*/
 void SCI_TX_ISR(void)

 {
  TXindex++;
  switch(ValidCommandChar)
   {
   case POS_REQ_COM_CHAR:
     *SCSR_Add     = *SCSR_Add;            /* Flag TDRE is cleared       */
     *SCDR_Add     = PosAckBuff[TXindex];  /* normal transmission        */
     if(TXindex >= (POS_ACK_STRING_LENGTH - 1))  /* last char of sequence */
      {
      DisableTXbuffEmptyInt();
      }
     break;
   case SER_REQ_COM_CHAR:
    *SCSR_Add     = *SCSR_Add;            /* Flag TDRE is cleared        */
    *SCDR_Add     = SerAckBuff[TXindex];  /* normal transmission         */
    if(TXindex >= (SER_ACK_STRING_LENGTH - 1))/* last char of sequence ?  */
     {
      DisableTXbuffEmptyInt();
     }
    break;
   default:
```

```
    printf("Unknown type of request for TX in SCI_TX_ISR()\r");
    break;
  } /* end switch()                                                              */
} /* end ISR to TX                                                               */

/*-----------------------------------------------------------------------------*/
/*                           ISR for RX                                        */
/*-----------------------------------------------------------------------------*/
void SCI_RX_ISR(void)

{
 SaveReg();  /* Read SCDR & SCSR --> clear RDRF bit                            */
#ifdef TEST0
 putchar((int)WorkRXdata);
#endif
 RXerror();  /* Check for Reception Errors                                     */
 if(     ( Overrun      == FALSE)
     && ( NoiseFraming == FALSE)) /* we are NOT presently rxing a string?*/
  {
  if(  (WorkRXdata == START_CHAR)
     &&(RXstream == FALSE)) /* and the first character is the start char */
   {
   RXstream = TRUE;                /*set the rxing stream semaphore         */
   RXindex = START_CHAR_INDEX;
                                /* start char is NOT put into string buffer */
   }
  else if (RXstream == TRUE)
   {
   CharRX();  /* in string, after start char, store character              */
  }
 }
 else
  RXstream = FALSE;
}
```

```
/**************************************************************************
/*                     George Mason University                             *
/*          Department of Electrical and Computer Engineering             *
/*                                                                        *
/* File Name:  FAMELIB.C                                                  *
/*                                                                        *
/*  Authors: Bertina Ho-Mock-Qai, Darrell Duane,  Ken Hintz               *
/*  Update History:  Version 1.0, February 23, 1992                       *
/**************************************************************************
/* Function called after a complete servo command sequence is received.   *
/* transmits 'SAs'                                                        *
/**************************************************************************
void FillSerAckBuff(void)

{
 SerAckBuff[START_CHAR_INDEX]           = START_CHAR;
 SerAckBuff[COM_CHAR_INDEX]             = SER_ACK_COM_CHAR;
 SerAckBuff[SER_ACK_STRING_LENGTH - 1]  = STOP_CHAR;
}


/**************************************************************************
/*      Checks for overrun, framing, noise error                          *
/**************************************************************************
void RXerror(void)

{
 Overrun              = FALSE;
 NoiseFraming         = FALSE;
 if(    (GetBit_Char(WorkSCSR,1)!=0)
      ||(GetBit_Char(WorkSCSR,2)!=0))
   {
   NoiseFraming = TRUE;
   }
 if(GetBit_Char(WorkSCSR,3) != 0)
   {
   Overrun=TRUE;
   }
}  /* end of function RXerror() */


/**************************************************************************
/*  Save the SCI receive buffer and the flag register of the SCI          *
/*  It is the first operation performed when a RX interrupt occurs        */
/**************************************************************************/
void SaveReg(void)  /* This function clears the RDRF flag of the SCSR     */

{
 WorkRXdata  = *SCDR_Add;
 WorkSCSR    = *SCSR_Add;
 WorkRXdata  = *SCDR_Add;    /* to clear interrupt                        */
 asm("int2 CLI\n");   /* enable interrupt to allow OCs to operate         */
}
```

```
/*************************************************************************.
/* Concatenates 2 characters RXed into an integer value                  .
/*************************************************************************.
int Concat_Int(unsigned char MSbits,unsigned char LSbits)

{
 MSBits = MSbits;
 LSBits = LSbits;
 concat_int();        /* assembly language routine in file concat.s11      */
 return Concat_2B;
}


/*************************************************************************
/*      This routine clears to 0 a given bit of a given byte.            *
/*      This byte is pointed to by pointer                               *
/*************************************************************************
void ClearBit(unsigned char *pointer,int NumBit)

{
 switch(NumBit)
  {
  case 0: *pointer = *pointer & MASK0;
         break;
  case 1: *pointer = *pointer & MASK1;
         break;
  case 2: *pointer = *pointer & MASK2;
         break;
  case 3: *pointer = *pointer & MASK3;
         break;
  case 4: *pointer = *pointer & MASK4;
         break;
  case 5: *pointer = *pointer & MASK5;
         break;
  case 6: *pointer = *pointer & MASK6;
         break;
  case 7: *pointer = *pointer & MASK7;
         break;
  }
}


/*************************************************************************
/*  This routine clears to 0 a bit of a given flag register (TFLG1 or    */
/*  TFLG2).  To do so, a 1 must be written to the bit to be cleared.     */
/*  (see HC11 manual)  It was verified that the assembly language        */
/*  translation of this function uses the BCLR instruction as prescribed */
/*  in section 10.2.4 of the reference manual                            */
/*************************************************************************/
void ClearFlag(unsigned char *pointer,int NumBit)

{
 switch(NumBit)
  {
  case 0: *pointer = CMASK0;
         break;
  case 1: *pointer = CMASK1;
         break;
  case 2: *pointer = CMASK2;
         break;
  case 3: *pointer = CMASK3;
```

```
        break;
  case 4: *pointer = CMASK4;
        break;
  case 5: *pointer = CMASK5;
        break;
  case 6: *pointer = CMASK6;
        break;
  case 7: *pointer = CMASK7;
        break;
  }
}

/***********************************************************************
/*      This routine sets to 1 a given bit of a given byte.            *
/*      This byte is pointed to by a pointer.                          *
/***********************************************************************/
void SetBit(unsigned char *pointer,int NumBit)

{
 switch(NumBit)
  {
  case 0: *pointer = *pointer | CMASK0;
        break;
  case 1: *pointer = *pointer | CMASK1;
        break;
  case 2: *pointer = *pointer | CMASK2;
        break;
  case 3: *pointer = *pointer | CMASK3;
        break;
  case 4: *pointer = *pointer | CMASK4;
        break;
  case 5: *pointer = *pointer | CMASK5;
        break;
  case 6: *pointer = *pointer | CMASK6;
        break;
  case 7: *pointer = *pointer | CMASK7;
        break;
  default: printf("Invalid Request to change bit = %d",NumBit);
  }
}

/************************************************************************/
/*      This routine allows to test the value of a given bit of a byte.  */
/*      This byte is pointed to by pointer.                              */
/************************************************************************/
unsigned char GetBit(unsigned char *pointer,int NumBit)

{
unsigned char BitResult;

 switch(NumBit)
  {
  case 0: BitResult=(*pointer & CMASK0);
        break;
  case 1: BitResult=(*pointer & CMASK1);
        break;
  case 2: BitResult=(*pointer & CMASK2);
        break;
  case 3: BitResult=(*pointer & CMASK3);
```

```
        break;
  case 4: BitResult=(*pointer & CMASK4);
        break;
  case 5: BitResult=(*pointer & CMASK5);
        break;
  case 6: BitResult=(*pointer & CMASK6);
        break;
  case 7: BitResult=(*pointer & CMASK7);
        break;
  }
 return(BitResult);
}


/*************************************************************************
/*      This routine allows to test the value of a given bit of a byte.  *
/*************************************************************************
unsigned char GetBit_Char(unsigned char reg,int NumBit)

{
unsigned char BitResult;

 switch(NumBit)
  {
  case 0: BitResult=(reg & CMASK0);
        break;
  case 1: BitResult=(reg & CMASK1);
        break;
  case 2: BitResult=(reg & CMASK2);
        break;
  case 3: BitResult=(reg & CMASK3);
        break;
  case 4: BitResult=(reg & CMASK4);
        break;
  case 5: BitResult=(reg & CMASK5);
        break;
  case 6: BitResult=(reg & CMASK6);
        break;
  case 7: BitResult=(reg & CMASK7);
        break;
  }
 return(BitResult);
}


/*************************************************************************/
/* Cuts an integer into 2 bytes & returns the values in MSbits and LSBits*/
/*************************************************************************/
void Split_Int(int Data_To_Split)

{
 Int_To_Split = Data_To_Split;
 split_Int();
}
```

```
/**************************************************************************
/*        This function calculates the checksum of sequences              *
/* ignores 0th, last, and (last - 1) elements of array                    *
/* i.e., it ignores the start, checksum, and stop characters              *
/**************************************************************************
unsigned char Checksum(int stringlength, unsigned char CheckArray[])

{
unsigned char ChecksumResult = 0;
unsigned int sum             = 0;
int i;

 for(i = 1; i < stringlength - 2; i++)
  sum = sum + CheckArray[i];
 ChecksumResult = (unsigned char)sum;
 return ChecksumResult;
}  /* end checksum function */
```

6.2. Appendix II: MC68HC11 Assembly Source Code

```
*-----------------------------------------------------------------*
*     Function to split an integer into 2 bytes MSBits LSBits      *
*-----------------------------------------------------------------*


      import MSBits            * Most significant byte result
      import LSBits            * Least significant byte result
      import Int_To_Split      * Integer to split

      section .text

split_int:
      ldaa Int_To_Split
      staa MSBits
      ldaa Int_To_Split+1
      staa LSBits
      rts
      end


*-----------------------------------------------------------------*
*     Function that concats 2 bytes into a 16 bit integer         *
*-----------------------------------------------------------------*


      import MSBits            * Most significant bits of the integer
      import LSBits            * Least significant bits of the integer
      import Concat_2B  * result of the concatenation

      section .text

concat_int:
      ldaa MSBits
      staa Concat_2B
      ldaa LSBits
      staa Concat_2B+1
      rts
      end
```

6.3. Appendix III: Linker Command File

```
/*-------------------------------------------------------------------*
/* File:  fame32-3.ld                                                *
/*    vers=1.0                                                       *
/*                                                                   *
/*    Configuration is a Motorola MC68HC11EVB evaluation board.      *
/*    This is designed for programs that are used WITH the           *
/*    BUFFALO monitor.  The executable code and initialized data will *
/*      placed in an 8Kx8 RAM at 0xC000.                             *
/*                                                                   *
/*   Modification history:                                           *
/*     KJH:  for 32kbyte SRAM 1/29/92                                *
/*-------------------------------------------------------------------*

set H11RAM = 0x00;                  /* page containing 68HC11 base page *
set H11REG = 0x01;                  /* page containing 68HC11 registers *

set H11VECSIZE = 42;                /* size of 68HC11 vectors          *
set H11RAMREG = (H11RAM<<4)|H11REG;/*mask for setting H11INIT in start.s *
set H11REGORG = (H11REG<<12)     /* origin of 68HC11 registers         */
set H11VECORG = 0x10000-H11VECSIZE;  /* origin of 68HC11 vectors         */

section .base bss origin 0 maxsize = 0x34 = .base;  /* uninitialized base page
storage    */
section .text1 origin 0x1800 maxsize (0x4000 - 0x1800) = .text;
section .text2 origin 0x6000 maxsize = 8192 = .text;

section .bss bss origin endof (.text2) bss comms;/* followed by uninit
storage*/
section .data origin endof (.bss);          /* followed by data         */
section .const origin endof(.data);         /* followed by constants    */
section .strings origin endof(.const);      /* followed by strings      *
section .init origin endof(.strings);       /* data to be copied to RAM *
section .heap bss origin endof (.init);     /* followed by heap         */
section .stack bss origin 0xd000 minsize (512);

/*-------------------------------------------------------------------*/
/* The following line can be used to copy data from ROM into RAM     */
/* substitute in place of the section .data line above where the     *
/* underscore is replaced with the address of the ROM where the data *
/* is to be stored                                                   *
/*                                                                   *
/* section .data origin ______ copiedfrom .init = .data;             *
/*-------------------------------------------------------------------*

/*-------------------------------------------------------------------*/
/* Sections particular to the registers and vectors in the EVB.      */
/* Used by the assembler to locate the registers and interrupt vectors */
/* at their proper location in memory since I/O is memory mapped.    */
/*-------------------------------------------------------------------*/

section .H11REGS bss origin H11REGORG;
section .H11VEC data origin H11VECORG maxsize H11VECSIZE;

/*-------------------------------------------------------------------*/
/* Checks for sections which exceed the limits of RAM on the EVB     *
/* If the optional RAM is installed, then modify the limits accordingly */
/*-------------------------------------------------------------------*/
```

```
check endof(.heap)   >= 0x7fff  fatal "Code area too large";
check sizeof(.base) > 0x34 fatal "Base page too large";

/*-----------------------------------------------------------------------*/
/* Initializes some values which are used by the initialization code to  */
/* initialize various areas of RAM                                        */
/*-----------------------------------------------------------------------*/

set _ramstart = startof(.bss);
set _ramend = endof(.bss);
set _heapstart = startof(.heap);
set _heapend = endof(.heap);
set _stackstart = startof(.stack);
set _stackend = endof(.stack);
set _initstart = startof(.init);
set _initend = endof(.init);
set _datastart = startof(.data);
set _dataend = endof(.data);

check _stackend - _stackstart < 256 fatal "Stack too small";

/*-----------------------------------------------------------------------*/
/* These object files replace functions in the library that are unique   */
/* to the Buffalo monitor version                                         */
/*                                                                        */
/* kjhstart.o, kjhiob.o and kjhbuff.o are recompiled versions of          */
/* (kjh)start.s, (kjh)iob.s, and (kjh)buffalo.s which allows using the    */
/* Buffalo monitor routines for I/O                                       */
/*                                                                        */
/* printf.o has been compiled to use the BUFFALO I/O Calls and should     */
/* not be included unless necessary since it takes up quite a bit of     */
/* space.                                                                 */
/*                                                                        */
/* ofmt.o has been compiled with the #define FLOATS  and LONGS omitted    */
/* to reduced size of code.                                               */
/*-----------------------------------------------------------------------*/

'c:\introl\kjh\kjhstart.o'    /* start up routine                  */
'c:\introl\kjh\kjhiob.o'        /* buffalo input output calls          */
'c:\introl\kjh\kjhbuff.o'
'c:\introl\dd8\concat.o'  /* puts two bytes RXed together to form an Int */
'c:\introl\dd8\split.o'     /* splits an Int into two bytes for TX to PC */

readline;                 /* read the command line              */

/*-----------------------------------------------------------------------*/
/* inclusion of the standard libraries to resolve external references    */
/*-----------------------------------------------------------------------*/

-lc                       /* use the C library                  */
-lcio                     /* C i/o library              */
-lm                       /* math library                       */
-lgen                     /* general library                    */
```

```
/*---------------------------------------------------------------------.
/* File:  fcal32-3.ld                                                  .
/*        vers=1.0                                                     .
/*                                                                     .
/*        Configuration is a Motorola MC68HC11EVB evaluation board.    .
/*        This is designed for programs that are used WITH the         .
/*        BUFFALO monitor.  The executable code and initialized data will .
/*        placed in an 8Kx8 RAM at 0xC000.                             .
/*                                                                     *.
/*    Modification history:                                            *.
/*      KJH:  for 32kbyte SRAM 1/29/92                                 */
/*      DD:   put jump commands for ISRs dir in locations at $0000 2/6/92 */
/*---------------------------------------------------------------------*/

set H11RAM = 0x00;                   /* page containing 68HC11 base page */
set H11REG = 0x01;                   /* page containing 68HC11 registers *

set H11VECSIZE = 42;                 /* size of 68HC11 vectors           *.
set H11RAMREG = (H11RAM<<4)|H11REG;/*mask for setting H11INIT in start.s */
set H11REGORG = (H11REG<<12)       /* origin of 68HC11 registers        *
set H11VECORG = 0x10000-H11VECSIZE;  /* origin of 68HC11 vectors         *.

set CAL_START = 0x7f00;               /* start of calibration values     */
section .base bss origin 0 maxsize = 0x34 = .base;  /* uninitialized base page
storage     */

section .ivt origin 0x00c4 = .ivt;   /* Begin Interrupt Vector Table (SCI is
first) at 0x00c4 */
section .text1 origin 0xc000 maxsize 8192 = .text;
section .text2 origin 0x1800 maxsize (CAL_START - 0x1800) = .text;

section .bss bss origin endof (.text2) bss comms;/* followed by uninit
storage*/
section .data origin endof (.bss);          /* followed by data            *
section .const origin endof(.data);         /* followed by constants       *.
section .strings origin endof(.const);      /* followed by strings         *.
section .init origin endof(.strings);       /* data to be copied to RAM    */
section .heap bss origin endof (.init);     /* followed by heap            */
section .stack bss origin (CAL_START - 512) minsize 512;

/*---------------------------------------------------------------------*/
/* The following line can be used to copy data from ROM into RAM       */
/* substitute in place of the section .data line above where the       */
/* underscore is replaced with the address of the ROM where the data   */
/* is to be stored                                                     */
/*                                                                     */
/* section .data origin ______ copiedfrom .init = .data;               */
/*---------------------------------------------------------------------*/

/*---------------------------------------------------------------------*/
/* Sections particular to the registers and vectors in the EVB.        */
/* Used by the assembler to locate the registers and interrupt vectors */
/* at their proper location in memory since I/O is memory mapped.      */
/*---------------------------------------------------------------------*/

section .H11REGS bss origin H11REGORG;
section .H11VEC data origin H11VECORG maxsize H11VECSIZE;

/*---------------------------------------------------------------------*/
/* Checks for sections which exceed the limits of RAM on the EVB       */
```

```
/* If the optional RAM is installed, then modify the limits accordingly  *
/*---------------------------------------------------------------------*

check endof(.stack)   > CAL_START  fatal "Code area too large";
check endof(.base) > 0x34 fatal "Base page too large";

/*---------------------------------------------------------------------*
/* Initializes some values which are used by the initialization code to  *
/* initialize various areas of RAM                                        *
/*---------------------------------------------------------------------*

set _ramstart = startof(.bss);
set _ramend = endof(.bss);
set _heapstart = startof(.heap);
set _heapend = endof(.heap);
set _stackstart = startof(.stack);
set _stackend = endof(.stack);
set _initstart = startof(.init);
set _initend = endof(.init);
set _datastart = startof(.data);
set _dataend = endof(.data);

check _stackend - _stackstart < 256 fatal "Stack too small";

/*---------------------------------------------------------------------*
/* These object files replace functions in the library that are unique  */
/* to the Buffalo monitor version                                       */
/*                                                                      */
/* kjhstart.o, kjhiob.o and kjhbuff.o are recompiled versions of        */
/* (kjh)start.s, (kjh)iob.s, and (kjh)buffalo.s which allows using the  */
/* Buffalo monitor routines for I/O                                     */
/*                                                                      *
/* printf.o has been compiled to use the BUFFALO I/O Calls and should   *
/* not be included unless necessary since it takes up quite a bit of    *
/* space.                                                               *
/*                                                                      *
/* ofmt.o has been compiled with the #define FLOATS  and LONGS omitted  *
/* to reduced size of code.                                             *
/*---------------------------------------------------------------------*

'c:\introl\kjh\kjhstart.o'      /* start up routine                    *
'c:\introl\kjh\kjhiob.o'        /* buffalo input output calls          *
'c:\introl\kjh\kjhbuff.o'
'c:\introl\dd8\splitdb.o'     /* splits an Int into four bytes for saving to
EEPROM */

readline;                       /* read the command line               */

/*---------------------------------------------------------------------*/
/* inclusion of the standard libraries to resolve external references   */
/*---------------------------------------------------------------------*/

-lc                             /* use the C library                   */
-lcio                           /* C i/o library                       */
-lm                             /* math library                        */
-lgen                           /* general library                     */
```

6.4. Appendix IV: PC (Workstation) Software

```
/*************************************************************************.
/*                        George Mason University                        .
/*          Department of Electrical and Computer Engineering            .
/*                                                                       .
/* File Name:  FAMEpc.C                                                 .
/*                                                                       .
/*  Authors: Bertina Ho-Mock-Qai, Darrell Duane,  Ken Hintz              .
/*  Update History:  Version 1.0, February 23, 1992                      .
/*                                                                       .
/*                                                                       .
/*************************************************************************.
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <float.h>
#include "famedef.h"
#include "pcdef.h"

static void (interrupt far *oldserialint)(void);
static void interrupt far newserialint(void);

void main()
{ unsigned char MenuChoice[3];
int ContinueIt,i,j,delay;   /* used to allow for continuous operation */
  SERIALPORT=1;           /* set port to com 1 */
  disable();              /* General interrupt mask */
  DisablePC_TXint();      /* Local TX mask */
  clrscr();
  InitSerialPort();       /* Initialize the UART: baud, port...*/
  InitTXparm();           /* Initialize the TX parameters */
  InitRXparm();           /* Initialize the RX parameters */
  DisablePC_RXint();
  enable();           /* General interrupt mask */
  ContinueIt=TRUE;
  NewData=FALSE;
  delaytime=5;            /* set delay between TX equal to 5 milliseconds */

while(ContinueIt)
  {

  printf("\n            Fast Adaptive Maneuvering Experiment -- PC
Interface\n\n");
  printf(" Do you want to send (P)osition request, (R)epetitive position
request, \n");
  printf(" (S)ervo control values, (D)efault servo control values, \n");
  printf(" r(E)petitively send variety of servo control values, \n");
  printf(" (I)nteractively send position requests & a variety of servo control
values, \n");
  printf(" (V)ary servo settings repetitively with keyboard, set (T)ime
in\n");
  printf(" milliseconds to delay between transmissions, (C)hange Com port or
(Q)uit?\n");
  scanf("%1s", MenuChoice);
  switch(toupper(MenuChoice[0]))
   {
   case 'P':
      PosOrSer='P';
      ClearRXbuffer();
```

```
        TXfirst();    /* Transmit first char of the PosReqBuff */
        AwaitAck();
        DisplayData();
        break;

    case 'R':
        PosOrSer='P';
        clrscr();
        printf("\n Repetitvely send position request to HC11.  Press any key to
exit.");
        WaitForEnter();
        while(!kbhit())
        {
            ClearRXbuffer();
            TXfirst();    /* Transmit first char of the PosReqBuff */
            AwaitAck();   /*  */
            clrscr();     /* clear screen */
            DisplayData();
        }
        break;

    case 'S':
        PosOrSer='S';
        EnterServoControlValues();
        FillSerReqBuff();
        ClearRXbuffer();
        TXfirst();    /* Transmit first char of SerReqBuff */
        AwaitAck();
        break;

    case 'D':
        PosOrSer='S'; /* servo control values desired */
        DefaultServoControlValues();
        FillSerReqBuff();
        ClearRXbuffer();
        TXfirst();
        AwaitAck();
        break;

    case 'E':
        j=0;
        PosOrSer='S';
        clrscr();
        printf("\n Repetitvely send varying servo control request to HC11.  ");
        printf("\n Press any key to exit.");
        WaitForEnter();
        while(!kbhit())
        {
            VariableServoControlValues(j);
            clrscr();
            FillSerReqBuff();
            ClearRXbuffer();
            TXfirst();    /* Transmit first char of the PosReqBuff */
            AwaitAck();
            if(++j==11) j=0;

        }
        break;

    case 'I':
        j=0;
```

```
      clrscr();
      printf("\n Interactively send position requests & varying servo control
requests \n");
      printf(" to HC11.  Press any key to exit.");
      WaitForEnter();
      while(!kbhit())
      {
         PosOrSer= 'P';
         ClearRXbuffer();
         TXfirst();    /* Transmit first char of the PosReqBuff */
         AwaitAck();   /*   */
         clrscr();
         DisplayData();
         PosOrSer='S';
         VariableServoControlValues(j);
         FillSerReqBuff();
         ClearRXbuffer();
         TXfirst();    /* Transmit first char of the PosReqBuff */
         AwaitAck();
         if(++j==11) j=0;
       }
       break;

   case 'V':
      PosOrSer='S'; /* TX servo control values */
      ContinueKeyboard=TRUE;
      DefaultServoControlValues();
      clrscr();
      printf("\n\n Type capital letters to increase & lowercase letters to
decrease \n");
      printf(" servo control settings\n");
      printf(" (T)hrottle, (A)ileron, (E)levator, (R)udder, (C)ollective\n ");
      printf(" Type Q to quit this session\n");
      WaitForEnter();
      while(ContinueKeyboard)
      {
      KeyboardServoControlValues();
      clrscr();
      FillSerReqBuff();
      ClearRXbuffer();
      TXfirst();
      AwaitAck();
      }
      break;

   case 'T':
      clrscr();
      printf("\n Delay time = %d ms.\n", delaytime);
      printf(" Delay time should be greater than 4 ms. \n");
      printf(" Enter new time in milliseconds to delay between transmissions:
");
      scanf("%d",&delaytime);
      break;

   case 'C':
      clrscr();
      BeginChangeComPort:
      printf("\n Change Com Port Value.  Com port = %d. \n",SERIALPORT);
      printf(" Enter new value for Com port (1 or 2): \n");
      scanf("%d", &SERIALPORT);
      if((SERIALPORT < 1) || (SERIALPORT > 2))
```

```
      { printf(" Invalid Com port value = %d.  Re-enter\n", SERIALPORT);
        goto BeginChangeComPort;  }
        SERIALPORT=1;            /* set port to com 1 */
       disable();          /* General interrupt mask */
       InitSerialPort();        /* Initialize the UART: baud, port...*/
       InitTXparm();            /* Initialize the TX parameters */
       InitRXparm();            /* Initialize the RX parameters */
       enable();            /* General interrupt mask */
      break;

   case 'Q':
      RestoreOldISR();       /* Restore the old interrupt service routine */
      ContinueIt=FALSE;
            break;
   default:
       printf("Invalid key hit-- reenter \n\n");
    break;

    }   /* end switch() */
  } /* end while loop for ContinueIt == TRUE  */
}  /* end main routine */

/*******************************************************************/
/* this function awaits acknowledgement from the HC11 & displays an */
/* error message if it isn't received in time                       */
/*******************************************************************/

void AwaitAck(void)

{
  delay(delaytime);
  while(RXint_Enabled)
  {   if( (biostime(0) > (TXtime + ONE_BIOS_SECOND)) )
      { printf(" Timeout:  ACK not Received from HC11! \n");
      DisablePC_RXint();  /* set equal to true so that user screen becomes
available */
       }
  } /* end while RXint_Enabled */

}  /* end function AwaitAck() */

/**********************************************************************/
/* Function to initialize the UART, attach the new ISR, save the old ISR */
/**********************************************************************/
void InitSerialPort(void)
{
   initserial.serial_initial_bits.parity=  PARITY==2 ? 3 : PARITY ;
   initserial.serial_initial_bits.stopbits= STOPBITS-1;
   initserial.serial_initial_bits.wordlen= WORDLEN-5;
   initserial.serial_initial_bits.brk =0;
   initserial.serial_initial_bits.divlatch =1;

   outportb(LINECTL,initserial.serial_initial_char);

   outportb(DIVLSB,(char) ((115200L/BAUD) & 255));
   outportb(DIVMSB,(char) ((115200L/BAUD) >> 8));

   initserial.serial_initial_bits.divlatch = 0;
   outportb(LINECTL,initserial.serial_initial_char);

   initializeISR();
```

```
}

/*************************************************************************.
/* Initializes ISR for TX & RX over serial port of PC                     .
/*************************************************************************.
void initializeISR(void)
{

 oldserialint = getvect(SERIALINT);    /* save the old ISR address */
 setvect(SERIALINT,newserialint);    /* attach the new ISR to the vector *

 outportb(MODEMCTL,(inportb(MODEMCTL) & 0xEF | DTR | RTS | OUT2));

 outportb(PIC01, (inportb(PIC01) & SERIALIRQ));  /* enable  the 8259 inter *

 outportb(PIC00,EOI);

 inportb(RXDATA);
 inportb(INTIDENT);
 inportb(LINESTATUS);
 inport(MODEMSTATUS);

/* printf(" Serial port initialized.\n"); */
}

/* Restore the old ISR attached to the com that we have used */
void RestoreOldISR(void)
{
  setvect(SERIALINT,oldserialint);
  printf("Old ISR restored\n");
}


/**************************************************************************
/* Initialize parameters for TX to 68HC11 board                          */
/**************************************************************************
void InitTXparm(void)

{
      /* Buffer to request position values of the heliocopter */

  PosReqBuff[START_CHAR_INDEX]=START_CHAR;
  PosReqBuff[COM_CHAR_INDEX]=POS_REQ_COM_CHAR;
  PosReqBuff[POS_REQ_STRING_LENGTH-1]=STOP_CHAR;

  TXindex=0;
}

/**************************************************************************
/* Prompts user for values to request servo control on heliocopter.      *
/**************************************************************************
void FillSerReqBuff(void)
{

  SerReqBuff[START_CHAR_INDEX]=START_CHAR;
  SerReqBuff[COM_CHAR_INDEX]=SER_REQ_COM_CHAR;

      SerReqBuff[THROTTLE_MSB] =(unsigned char)(ServoValues[THROTTLE]>>8);   *
put msbits here */
      SerReqBuff[THROTTLE_LSB] = (unsigned char)ServoValues[THROTTLE];   /* put
lsbits here */
```

```
      printf("Throttle:     %4d\n",ServoValues[THROTTLE]);

      SerReqBuff[AILERON_MSB] =(unsigned char)(ServoValues[AILERON]>>8);   /*
put msbits here */
      SerReqBuff[AILERON_LSB] = (unsigned char)ServoValues[AILERON];    /* put
lsbits here */
      printf("Aileron:      %4d\n",ServoValues[AILERON]);

      SerReqBuff[ELEVATOR_MSB] =(unsigned char)(ServoValues[ELEVATOR]>>8);   /*
put msbits here */
      SerReqBuff[ELEVATOR_LSB] = (unsigned char)ServoValues[ELEVATOR];    /* put
lsbits here */
      printf("Elevator:     %4d\n",ServoValues[ELEVATOR]);

      SerReqBuff[RUDDER_MSB] =(unsigned char)(ServoValues[RUDDER]>>8);    /* put
msbits here */
      SerReqBuff[RUDDER_LSB] = (unsigned char)ServoValues[RUDDER];   /* put
lsbits here */
      printf("Rudder:       %4d\n",ServoValues[RUDDER]);

      SerReqBuff[COLLECTIVE_MSB] =(unsigned char)(ServoValues[COLLECTIVE]>>8);
/* put msbits here */
      SerReqBuff[COLLECTIVE_LSB] = (unsigned char)ServoValues[COLLECTIVE];   /*
put lsbits here */
      printf("Collective:   %4d\n",ServoValues[COLLECTIVE]);

  SerReqBuff[SER_REQ_STRING_LENGTH-2 ]=Checksum(SER_REQ_STRING_LENGTH,
SerReqBuff);  /* Calculate Checksum for servo control values */
  SerReqBuff[SER_REQ_STRING_LENGTH-1 ]=STOP_CHAR;

} /* end function FillSerReqBuff*/


/*****************************************************************************
/* Takes input from keyboard to determine servo control values.              *
/*****************************************************************************
void EnterServoControlValues(void)

{
   printf("\nEnter servo control value between 2000 & 4000 for   THROTTLE: ");
      scanf("%d", &ServoValues[THROTTLE]);

   printf("\nEnter servo control value between 2000 & 4000 for    AILERON: ");
      scanf("%d", &ServoValues[AILERON]);

   printf("\nEnter servo control value between 2000 & 4000 for   ELEVATOR: ");
      scanf("%d", &ServoValues[ELEVATOR]);

   printf("\nEnter servo control value between 2000 & 4000 for     RUDDER: ");
      scanf("%d", &ServoValues[RUDDER]);

   printf("\nEnter servo control value between 2000 & 4000 for COLLECTIVE: ");
      scanf("%d", &ServoValues[COLLECTIVE]);

} /* end function EnterServoControlValues()*/

void DefaultServoControlValues(void)

{

 printf("\nLoading Default Servo Control Values \n");
```

```
   ServoValues[THROTTLE]=THROTTLE_DEFAULT;
   ServoValues[AILERON]=AILERON_DEFAULT;
   ServoValues[ELEVATOR]=ELEVATOR_DEFAULT;
   ServoValues[RUDDER]=RUDDER_DEFAULT;
   ServoValues[COLLECTIVE]=COLLECTIVE_DEFAULT;

} /* end function DefaultServoControlValues */

void VariableServoControlValues(int multiplier)

{

 printf("\nLoading Variable Servo Control Value set # %d \n",multiplier);

   ServoValues[THROTTLE]=ONE_MS+(int)((multiplier * ONE_MS)/10);
   ServoValues[AILERON]=ONE_MS+(int)((multiplier * ONE_MS)/10);
   ServoValues[ELEVATOR]=ONE_MS+(int)((multiplier * ONE_MS)/10);
   ServoValues[RUDDER]=ONE_MS+(int)((multiplier * ONE_MS)/10);
   ServoValues[COLLECTIVE]=ONE_MS+(int)((multiplier * ONE_MS)/10);

} /* end function VariableServoControlValues */

void KeyboardServoControlValues(void)

{
    while(kbhit())
      {
      switch(getch())
        { case 'T':
            ServoValues[THROTTLE]+=SERVO_CHANGE_RATE;
          break;
          case 't':
            ServoValues[THROTTLE]-=SERVO_CHANGE_RATE;
          break;
          case 'A':
            ServoValues[AILERON]+=SERVO_CHANGE_RATE;
          break;
          case 'a':
            ServoValues[AILERON]-=SERVO_CHANGE_RATE;
          break;
          case 'E':
            ServoValues[ELEVATOR]+=SERVO_CHANGE_RATE;
          break;
          case 'e':
            ServoValues[ELEVATOR]-=SERVO_CHANGE_RATE;
          break;
          case 'R':
            ServoValues[RUDDER]+=SERVO_CHANGE_RATE;
          break;
          case 'r':
            ServoValues[RUDDER]-=SERVO_CHANGE_RATE;
          break;
          case 'C':
            ServoValues[COLLECTIVE]+=SERVO_CHANGE_RATE;
          break;
          case 'c':
            ServoValues[COLLECTIVE]-=SERVO_CHANGE_RATE;
          break;
          case 'Q':
          case 'q':
            ContinueKeyboard=FALSE;
```

```
          break;
          default:
          break;
        }  /* end switch() */
   } /* end while(kbhit()) */
} /* end function KeyboardServoControlValues() */


/***************************************************************************
/* Initiates TX Sequence to the HC11                                      *
/***************************************************************************
void TXfirst(void)
{
TXindex=1;  /* This will be the index that the ISR will use to TX the buffers
first value */
          /* This is not the TX ISR */

 while(getbit(inportb(LINESTATUS),5)==0) {putch('w'); putch('!'); }
               /* wait for TX to complete */
 switch(PosOrSer)
  {
    case 'P':
       outportb(TXDATA,PosReqBuff[START_CHAR_INDEX]);
       EnablePC_TXint();
       break;

    case 'S':
       outportb(TXDATA,SerReqBuff[START_CHAR_INDEX]);
       EnablePC_TXint();
       break;

    default:
      printf("Error Unknown Type of TX \n");
      break;

  }
TXtime=biostime(0);

}


/***************************************************************************
/*    initialized flags and semaphores for receiving data from HC11       */
/***************************************************************************
void InitRXparm(void)
{
  RXstream=FALSE;
  RXindex=0;
  outportb(LINECTL,(inportb(LINECTL)|0x80));
/*  printf("DLAB bit in LCR is set = 1"); */
/*  printf("LCR = 0x%x\n",inportb(LINECTL)); */   /* Line Control Register *
/*  printf("BAUD0 = 0x%x ",inportb(DIVLSB)); */
/*  printf("BAUD1 = 0x%x ",inportb(DIVMSB)); */
  outportb(LINECTL,(inportb(LINECTL)&0x7f));
/*   printf("DLAB bit in LCR is set = 0 ");     */
/*   printf("DATA = 0x%x\n", inportb(RXDATA)); */  /* Receive data value */
/*   printf("LCR = 0x%x\n", inportb(LINECTL)); */  /* Line Control Register *
/*   printf("MCR = 0x%x\n", inportb(MODEMCTL)); */ /* Modem Control Register
*/
/*   printf("IER = 0x%x\n", inportb(INTENABLE)); */ /* Interrupt Enable
Register */
```

```
/*   printf("LSR = 0x%x\n", inportb(LINESTATUS)); */  /* Line Status Register
*/
/*   printf("MSR = 0x%x\n", inportb(MODEMSTATUS)); */ /* Modem Status Register
Values */
/*   printf("IID = 0x%x\n", inportb(INTIDENT)); */ /* Interrupt Identification
& Causes */

}

/***********************************************************************
/* Clears RX buffer */
/***********************************************************************
void ClearRXbuffer(void)

{
char extra;

 while(getbit(inport(LINESTATUS),0))
   {    extra=inportb(RXDATA);
      printf("Cleared byte = 0x%x = '%c' from serial port RX
buffer.\n",(unsigned char)extra, extra);
    }
} /* end function ClearRXbuffer */

/***********************************************************************/
/* Checks for the noise and overrun errors after reception */
/***********************************************************************/
void RXerror(void)
{
 if ((getbit(WorkLinestat,3)!=0)||(getbit(WorkLinestat,2)!=0))
       NoiseFraming=TRUE;
 else
      { NoiseFraming=FALSE;
      if (getbit(WorkLinestat,1)!=0)
         if ((RXindex==(POS_ACK_STRING_LENGTH-1-1))&&(WorkRXdata==STOP_CHAR))
            Overrun=FALSE;
         else
            Overrun=TRUE;
      else
         Overrun=FALSE;
     }
}

/***********************************************************************/
/* Test if the received word is the start word                         */
/***********************************************************************/
void ResearchStartChar(void)
{  if (WorkRXdata==START_CHAR)
       {
        RXstream=TRUE;
        RXindex=START_CHAR_INDEX;
        /* putch('A'); putch('!');   A! start char received! */
       }
   else
    { putch('S'); putch('!'); } /* display S! for start char not received */
}

/***********************************************************************
/* Function to receive data from the HC11                              */
/***********************************************************************/
void CharRX(void)
```

```
{
  RXindex++;

  if(RXindex==    _CHAR_INDEX)    /* should this be the Command Character? */
      {  Work_ mandChar=WorkRXdata;
          switch(WorkCommandChar) {
            case POS_ACK_COM_CHAR:
            PosAckBuff[COM_CHAR_INDEX]=WorkRXdata;
            break;

            case SER_ACK_COM_CHAR:
            PosAckBuff[COM_CHAR_INDEX]=WorkRXdata;
            break;

            default:
            putch('U'); putch('!');
            ClearWorkVar();

            break;

         }  * end switch for Command Character */
     } /* end should this be the Command Character */

  else { /* this is not the command character */

      switch(WorkCommandChar)   {

       case POS_ACK_COM_CHAR:
           if( (RXindex > COM_CHAR_INDEX)
             &&(RXindex <= (POS_ACK_STRING_LENGTH-1-1)) )
             { PosAckBuff[RXindex]=WorkRXdata; /* put Position or Checksum
values into array to be un-concatenated */
             /*  putch('I');     print 'I' to the screen */
            }
           else
           if(RXindex == POS_ACK_STRING_LENGTH-1)
             { if(WorkRXdata == STOP_CHAR)
              { CompleteStream();
                DisablePC_RXint();  }
             else { putch('Y'); putch('!');  ClearWorkVar(); } }
           else
            { ClearWorkV  (); putch('Z'); putch('!'); }
            /* error out of synch */
          break;

        case SER_ACK_COM_CHAR:  /* is this a servo control acknowledgement? */
          if(RXindex==SER_ACK_STRING_LENGTH-1)  /* should this be the stop
char? */
            if(WorkRXdata==STOP_CHAR) /* is this the stop char? */
            { ClearWorkVar();
              /* putch('R'); putch('!'); Servo Control Acknowledgement RXed */
               }   /* disable rx inter */
            else  /* error */
            { putch('W'); putch('!');  ClearWorkVar(); }
          else
            { putch('X'); putch('!'); ClearWorkVar(); }
          break;

        default:
          printf("Work Control Word Not Recognized as RXed ");
          ClearWorkVar  ;
```

```
          break;
        } /* End of switch() */
      } /* End of else this shouldn't be the Command Character */

} /* end function CharRX */

/***************************************************************************
/*  Initialize semaphores for RXing a new string from the HC11.           *
/***************************************************************************
void ClearWorkVar(void)
{ int i;

  RXindex=0;
  RXstream=FALSE;
  DisablePC_RXint();
}


/***************************************************************************
/* Called upon RX of Stop char of Position acknowledgement sequence       *
/***************************************************************************
void CompleteStream(void)

{

 if( PosAckBuff[POS_ACK_STRING_LENGTH-1-1] == Checksum(POS_ACK_STRING_LENGTH,
PosAckBuff) ) /* Checks if the checksum is correct */
 {
   /* printf("\nStream complete\n");  */
   /* printf("command char = %x \n",WorkCommandChar);   */
  NewData=TRUE;
  }
 else
   { printf("Checksum Error: RXed Checksum= %x, Calculated Checksum = %x
\n",PosAckBuff[POS_ACK_STRING_LENGTH-1-1], Checksum(POS_ACK_STRING_LENGTH,
PosAckBuff));     }

 ClearWorkVar();}

/****************************************************************************
/*        This function calculates the checksum of sequences               *
/* ignores 0th, last, and (last - 1) elements of array                     *
/* i.e., it ignores the start, checksum, and stop characters
      */
/****************************************************************************
unsigned char Checksum(int stringlength, unsigned char CheckArray[])

{
unsigned char ChecksumResult = 0;
unsigned int sum             = 0;
int i;

 for(i = 1; i < stringlength - 2; i++)
  sum = sum + CheckArray[i];
 ChecksumResult = (unsigned char)sum;
 return ChecksumResult;
}  /* end checksum function */


/****************************************************************************/
```

```
/* concats 2 unsigned characters to an integer
 */
/**********************************************************************.
int Concat_Int(unsigned char MSbits,unsigned char LSbits)

{ unsigned int result;

   result=(int)MSbits;
   result=(result << 8);
   result=result+LSbits;
   return result;
}


/***********************************************************************
/* Concats Position values from the RX buffer & displays them to the screen *
/***********************************************************************
void DisplayData(void)
{ int i;

 clrscr();
 printf(" X=     %7.2f cm  \n", (float)Concat_Int(PosAckBuff[X_MSB],
PosAckBuff[X_LSB])/100 );
 printf(" Y=     %7.2f cm  \n", (float)Concat_Int(PosAckBuff[Y_MSB],
PosAckBuff[Y_LSB])/100 );
 printf(" Z=     %7.2f cm  \n", (float)Concat_Int(PosAckBuff[Z_MSB],
PosAckBuff[Z_LSB])/100 );

 printf(" Pitch= %7.2f deg \n", (float)Concat_Int(PosAckBuff[PITCH_MSB],
PosAckBuff[PITCH_LSB])/100 );
 printf(" Roll=  %7.2f deg \n", (float)Concat_Int(PosAckBuff[ROLL_MSB],
PosAckBuff[ROLL_LSB])/100 );
 printf(" Yaw=   %7.2f deg \n", (float)Concat_Int(PosAckBuff[YAW_MSB],
PosAckBuff[YAW_LSB])/100 );

/* printf(" X=     %7.2f cm  MSB: %2x  LSB: %2x\n",
(float)Concat_Int(PosAckBuff[X_MSB], PosAckBuff[X_LSB])/100,
PosAckBuff[X_MSB], PosAckBuff[X_LSB] );
   printf(" Y=     %7.2f cm  MSB: %2x  LSB: %2x\n",
(float)Concat_Int(PosAckBuff[Y_MSB], PosAckBuff[Y_LSB])/100,
PosAckBuff[Y_MSB], PosAckBuff[Y_LSB] );
   printf(" Z=     %7.2f cm  MSB: %2x  LSB: %2x\n",
(float)Concat_Int(PosAckBuff[Z_MSB], PosAckBuff[Z_LSB])/100,
PosAckBuff[Z_MSB], PosAckBuff[Z_LSB] );

   printf(" Pitch= %7.2f deg MSB: %2x  LSB: %2x\n",
(float)Concat_Int(PosAckBuff[PITCH_MSB], PosAckBuff[PITCH_LSB])/100,
PosAckBuff[PITCH_MSB], PosAckBuff[PITCH_LSB] );
   printf(" Roll=  %7.2f deg MSB: %2x  LSB: %2x\n",
(float)Concat_Int(PosAckBuff[ROLL_MSB], PosAckBuff[ROLL_LSB])/100,
PosAckBuff[ROLL_MSB], PosAckBuff[ROLL_LSB] );
   printf(" Yaw=   %7.2f deg MSB: %2x  LSB: %2x\n",
(float)Concat_Int(PosAckBuff[YAW_MSB], PosAckBuff[YAW_LSB])/100,
PosAckBuff[YAW_MSB],  PosAckBuff[YAW_LSB] );
*/
for(i=0; i<=POS_ACK_STRING_LENGTH-1 ; i++) PosAckBuff[i]=0;

}

/*----------------------------------------------------------------------*/
/*   ISR for transmitting to HC11                                       */
```

```
/*-------------------------------------------------------------------------*
void PC_TX_ISR(void)
{
 switch(PosOrSer)
  {
    case 'P':            /* We want to transmit the PosReq Buffer */
      while(getbit(inportb(LINESTATUS),5)==0) {putch('y'); putch(' ');}
      outportb(TXDATA,PosReqBuff[TXindex]);
      if(TXindex++==POS_REQ_STRING_LENGTH-1) { DisablePC_TXint();
EnablePC_RXint(); }
      break;

    case 'S':            /* We want to transmit the servo control parameters */
      while(getbit(inportb(LINESTATUS),5)==0) {putch('z'); putch(' ');}
      outportb(TXDATA,SerReqBuff[TXindex]);
      if(TXindex++==SER_REQ_STRING_LENGTH-1) { DisablePC_TXint();
EnablePC_RXint();  }
      break;

    default:
      printf("general error \n");
      break;
  } /* end switch() */
}


/*----------------------------------------------------------------------*/
/* Reception ISR                                                        */
/*----------------------------------------------------------------------*/
void PC_RX_ISR(void)
{ WorkRXdata=inportb(RXDATA);
  /* putch(WorkRXdata); putch('*'); display values received from HC11*/
  WorkLinestat=inportb(LINESTATUS);

  RXerror();

  if ((Overrun==FALSE)&&(NoiseFraming==FALSE))
       if (RXstream==FALSE)        /* we want the start char */
       { /* putch('f'); */  ResearchStartChar();  }
       else
       { /* putch('t'); */  CharRX();   }
  else
     ClearWorkVar();
}

/*----------------------------------------------------------------------*/
/* New communication interrupt service routine                          */
/*----------------------------------------------------------------------*/

static void interrupt far newserialint(void)
{ char identreg;

 identreg=inportb(INTIDENT);

  switch (identreg)
   {
   case 4:
       PC_RX_ISR();
       break;

   case 2:
```

```
        PC_TX_ISR();
        break;

      default:
        printf("default int\n");
        inportb(RXDATA);
        break;

    }
  outportb(PIC00,EOI);

}


/*************************************************************************
/* Pauses for user to read message on screen.                           *
/*************************************************************************

void WaitForEnter(void)
{
  printf("\n Press any key to begin.");
  while(!kbhit());

  getch();

} /* end function WaitForEnter() */


/*------------------------------------------------------------------------*
/*           File which contains some basic functions            */
/*------------------------------------------------------------------------*/
unsigned char clearbit(unsigned char reg,unsigned char NumBit)
{ unsigned char treg;

   switch(NumBit)
   {  case 0: treg = reg & MASK0;
             break;
      case 1: treg = reg & MASK1;
             break;
      case 2: treg = reg & MASK2;
             break;
      case 3: treg = reg & MASK3;
             break;
      case 4: treg = reg & MASK4;
             break;
      case 5: treg = reg & MASK5;
             break;
      case 6: treg = reg & MASK6;
             break;
      case 7: treg = reg & MASK7;
             break;
   }
  return(treg);
}

unsigned char setbit(unsigned char reg,unsigned char NumBit)
{ unsigned char treg;

  switch(NumBit)
   {  case 0: treg = reg | CMASK0;
             break;
```

```
      case 1: treg = reg | CMASK1;
              break;
      case 2: treg = reg | CMASK2;
              break;
      case 3: treg = reg | CMASK3;
              break;
      case 4: treg = reg | CMASK4;
              break;
      case 5: treg = reg | CMASK5;
              break;
      case 6: treg = reg | CMASK6;
              break;
      case 7: treg = reg | CMASK7;
              break;
    }
  return(treg);
}

unsigned char getbit(unsigned char reg,unsigned char NumBit)
{
 unsigned char BitResult;

  switch(NumBit)
   { case 0: BitResult=(reg & CMASK0);
             break;
     case 1: BitResult=(reg & CMASK1);
             break;
     case 2: BitResult=(reg & CMASK2);
             break;
     case 3: BitResult=(reg & CMASK3);
             break;
     case 4: BitResult=(reg & CMASK4);
             break;
     case 5: BitResult=(reg & CMASK5);
             break;
     case 6: BitResult=(reg & CMASK6);
             break;
     case 7: BitResult=(reg & CMASK7);
             break;
   }
  return(BitResult);
}
```

6.5. Appendix V: Mechanical Drawings

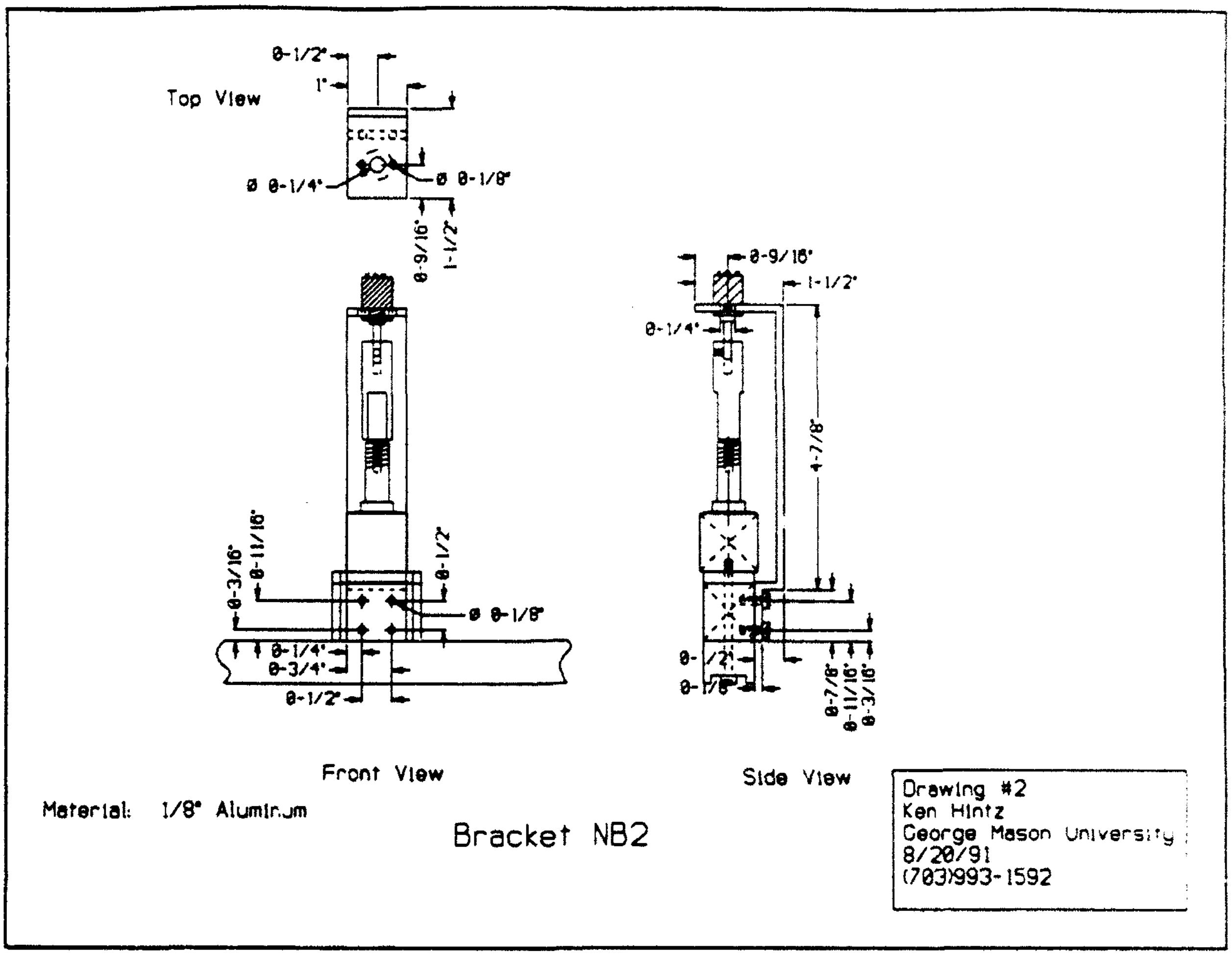

Figure 2 Bracket supporting potentiometer at center of base (H) of stand.

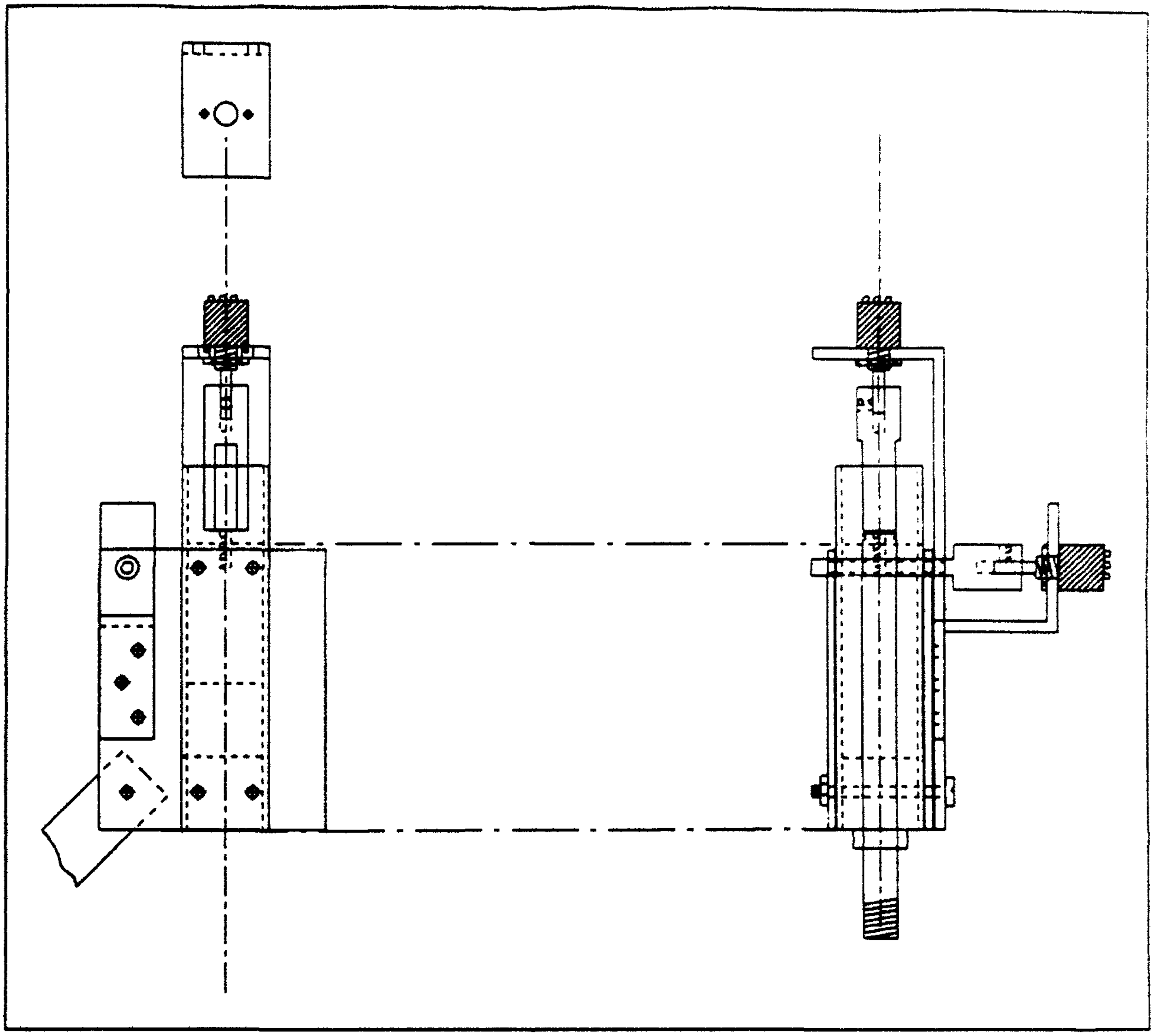

Figure 3 Relative location of brackets and potentiometers at middle joint.

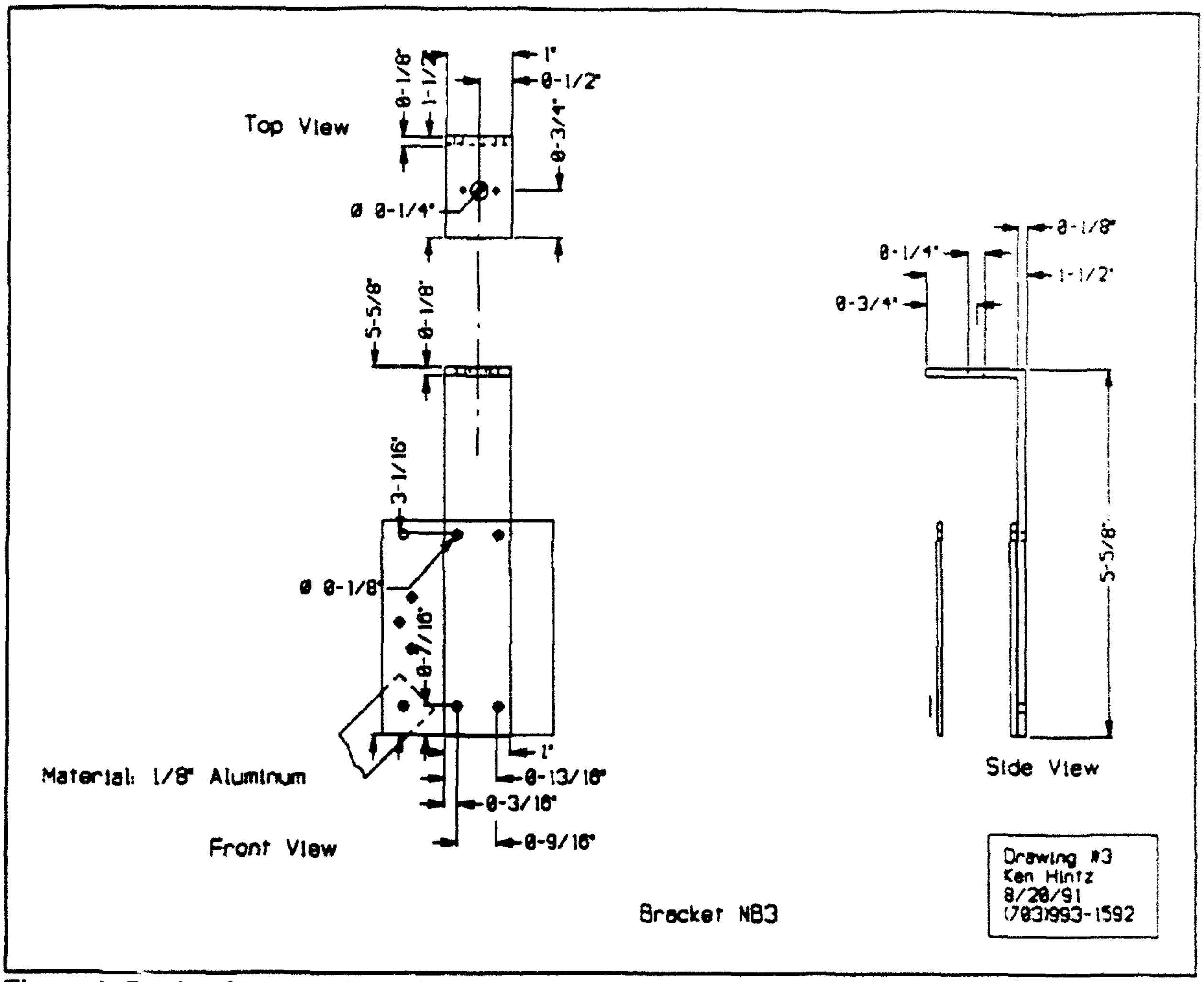

Figure 4 Bracket for supporting azimuth potentiometer at middle joint.

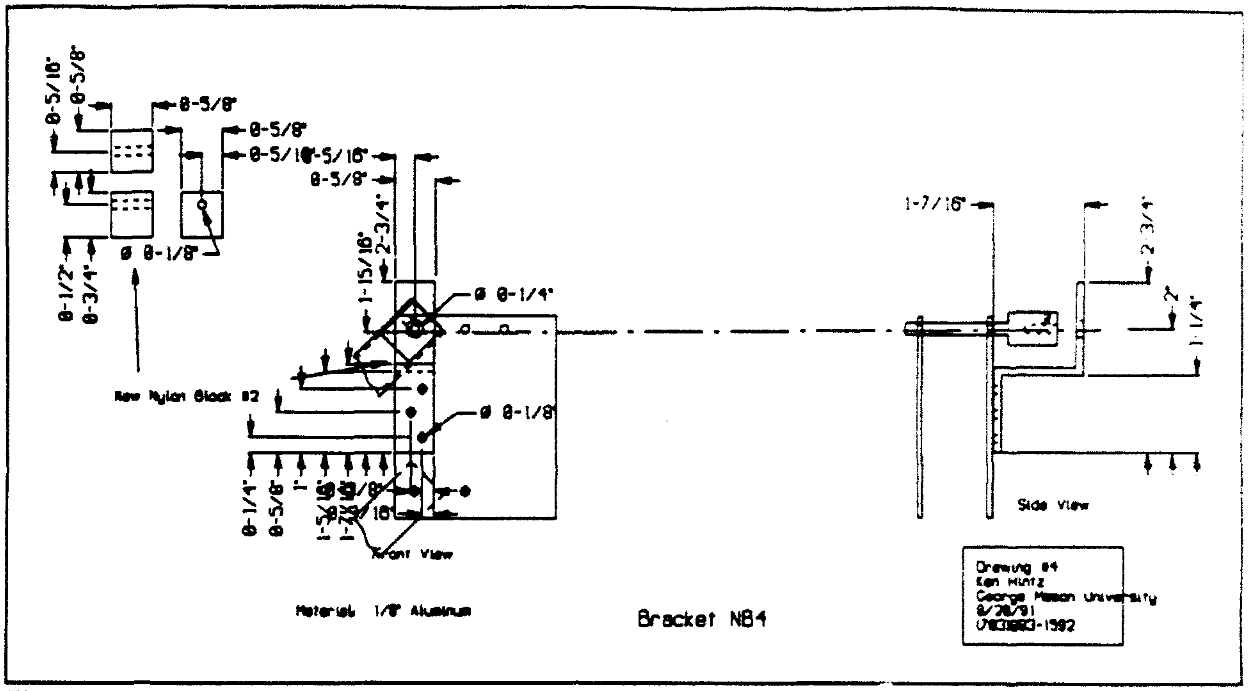

Figure 5 Bracket for supporting elevation potentiometer at middle joint.

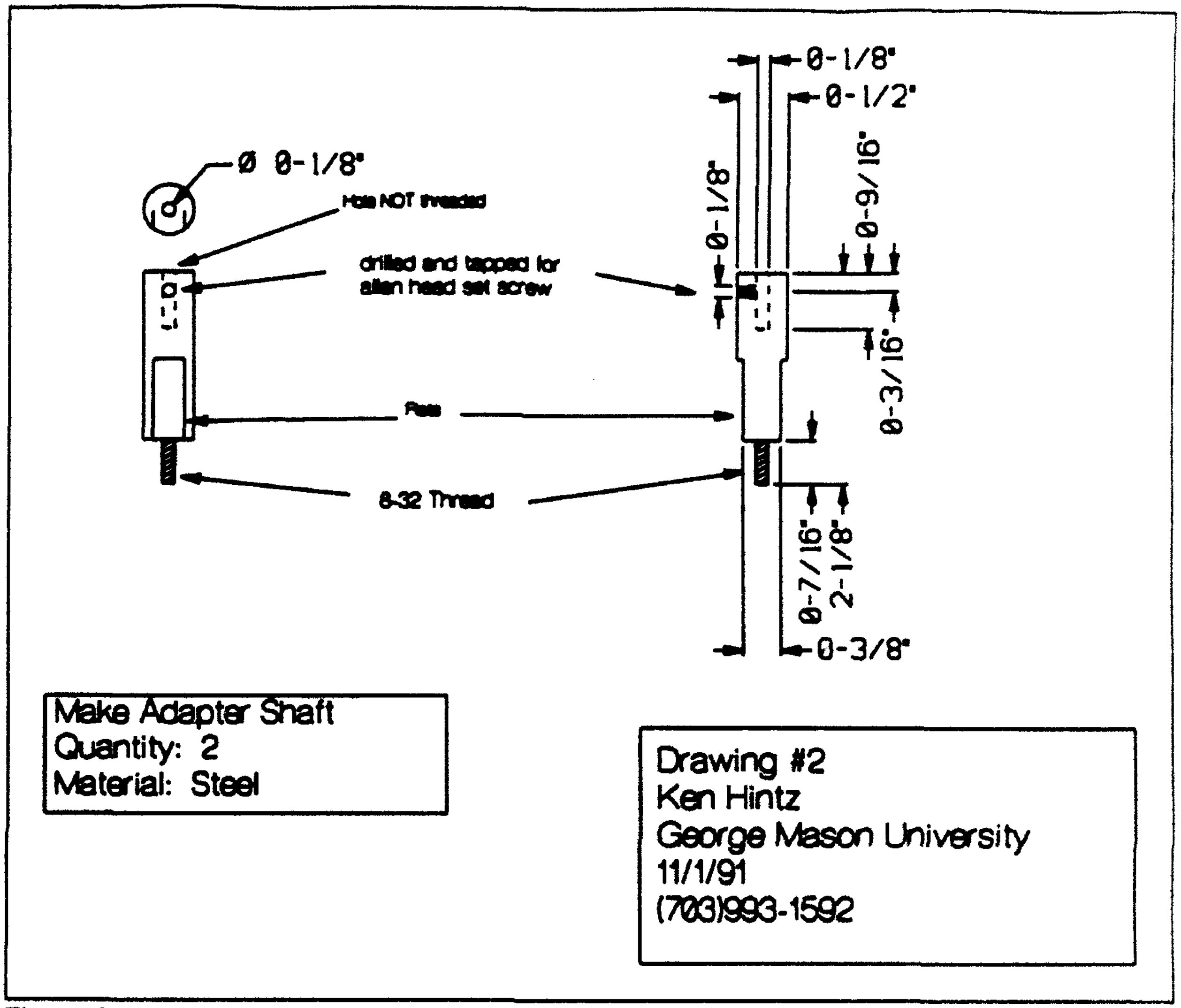

Figure 6 Adapter shaft to connect potentiometer to H-potentiometer and Azimuth-Potentiometer.

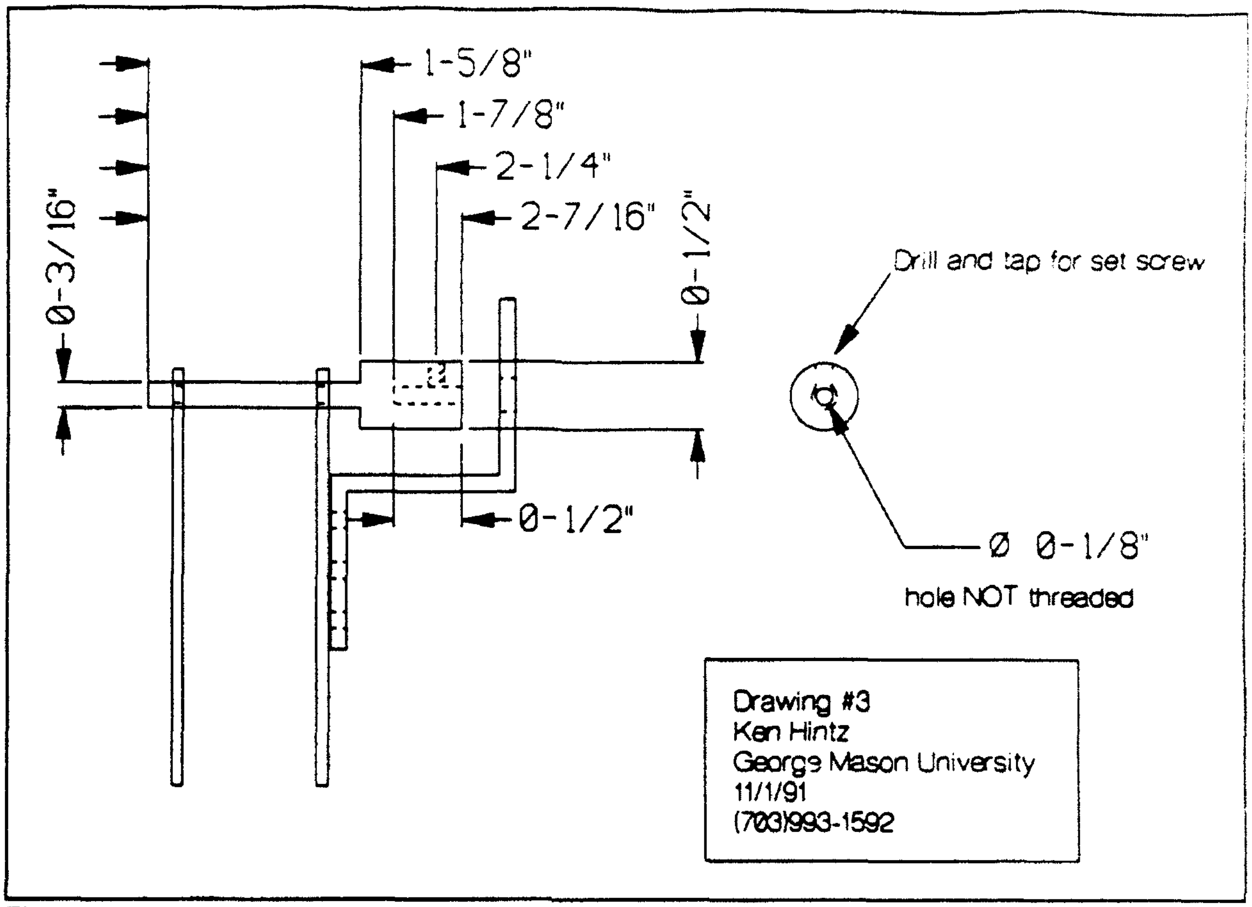

Figure 7 Adapter to connect elevation potentiometer to parallel elevation arms.

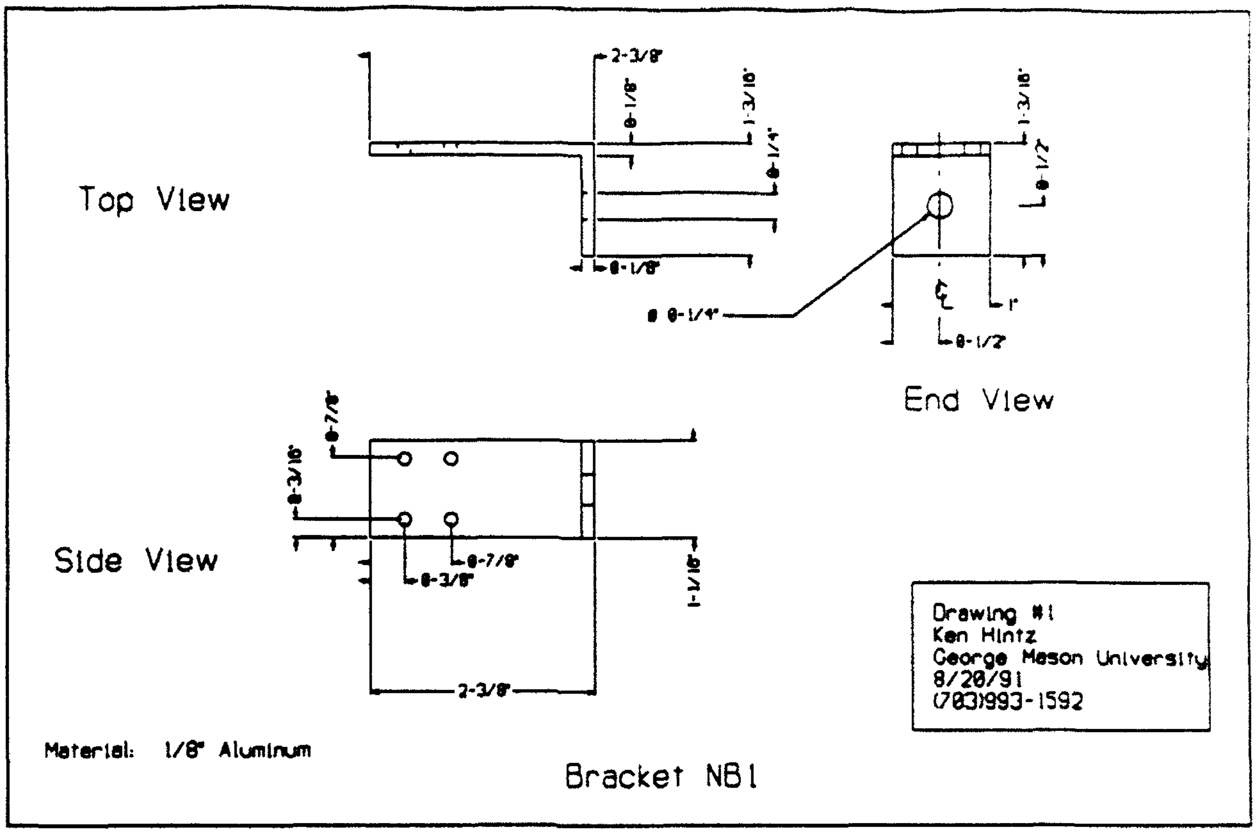

Figure 8 Bracket to support yaw potentiometer which connects directly to virtical helo support shaft.

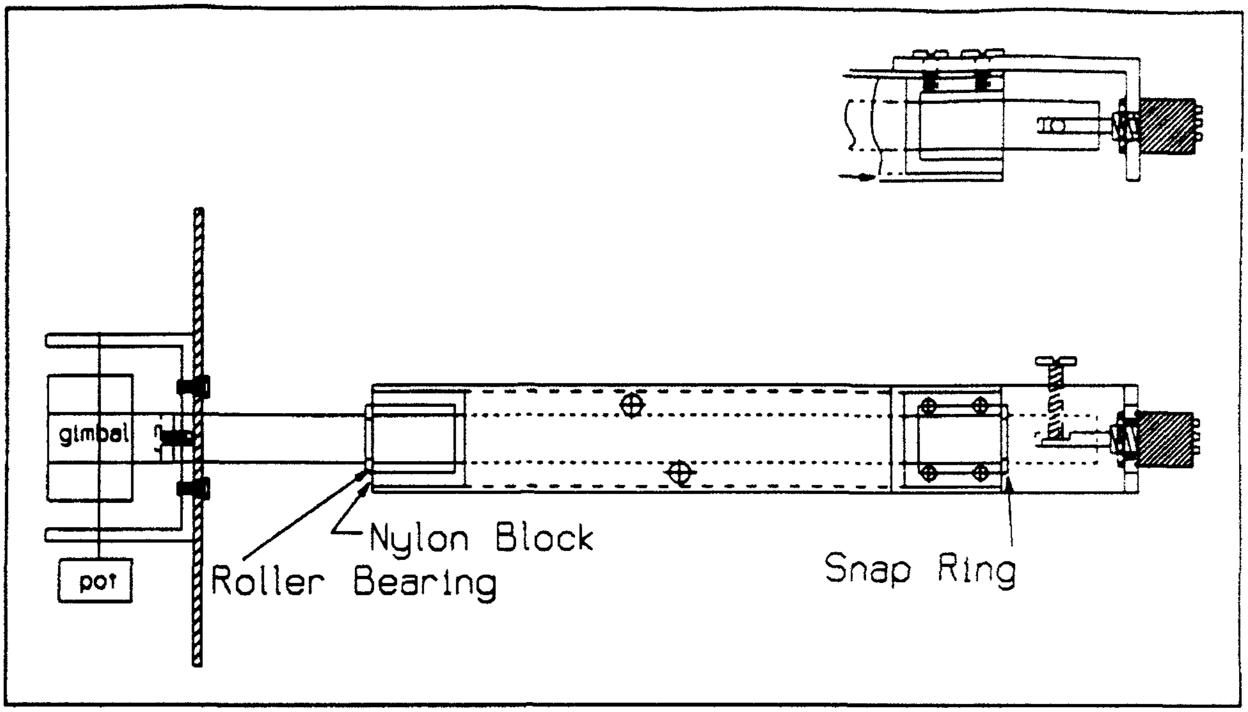

Figure 9 Relative location of support components at helicopter end of stand.

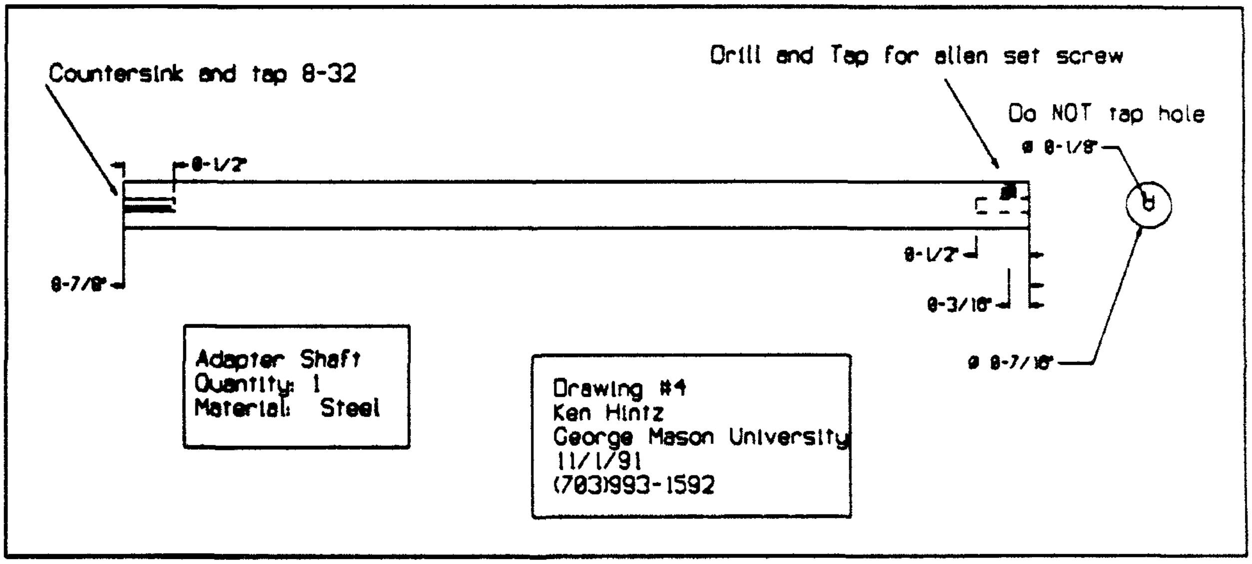

Figure 10 Shaft to support helicopter and connect to yaw potentiometer. Length could be extended to increase range of motion and still prevent tail rotor/boom strikes to stand.